НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

(повне найменування інституту, факультету)

Автоматизованих систем обробки інформації і управління

(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Розподілене сховище даних на основі Apache Spark*

Виконав: студент IV курсу, групи *ІП-63 Мамута Максим Дмитрович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. викладач, Олійник Ю.О.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

Доц. каф ТК, к.т.н., доц. Ткач М.М.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
10.06.2020 01:17:33 EEST

Дата звіту:
10.06.2020 01:30:06 EEST

ID перевірки:
1003919705

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Mamuta_bachelor_ip63

ID файлу: 1003934951 **Кількість сторінок:** 58 **Кількість слів:** 7822 **Кількість символів:** 58501 **Розмір файлу:** 6.23 MB

6.37% Схожість

Найбільша схожість: 5.15% з джерело бібліотеки. **ID файлу:** 1000037963

5.18% Схожість з Інтернет джерелами	6	Page 60
6.37% Текстові збіги по Бібліотеці акаунту	62	Page 60

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	1

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6318. 045430.01.81	Пояснювальна записка	71	
3	А4	КПІ.ІП-6318. 045430.02.13	Опис програми	23	
4	А4	КПІ.ІП-6318. 045430.03.91	Технічне завдання	12	
5	А4	КПІ.ІП-6318. 045430.04.51	Програма та методика тестування	7	
6	А4	КПІ.ІП-6318. 045430.05.34	Керівництво користувача	8	
7	А3	КПІ.ІП-6318. 045430.06.99 СС	Схема структурна варіантів використання	1	
8	А3	КПІ.ІП-6318. 045430.06.99 СС	Схема структурна бізнес процесів	2	
9	А3	КПІ.ІП-6318. 045430.06.99 КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6318.045430.00.90		
Розробн.		Мамута М.Д.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Олійник Ю.О.					
Консульт						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-63	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Мамуті Максиму Дмитровичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Розподілене сховище даних на основі Apache Spark*»

керівник проєкту *Олійник Юрій Олександрович*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту

Технічне завдання

4. Зміст пояснювальної записки

1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог

2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення

3) Аналіз якості та тестування програмного забезпечення

4) Впровадження та супровід програмного забезпечення

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема структурна бізнес процесів

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*28.02.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*07.03.2020*	
3.	*Постановка та формалізація задачі*	*12.03. 2020*	
4.	*Аналіз вимог до програмного забезпечення*	*17.03. 2020*	
5.	*Алгоритмізація задачі*	*25.03. 2020*	
6.	*Моделювання програмного забезпечення*	*29.03. 2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*04.04. 2020*	
8.	*Розробка архітектури програмного забезпечення*	*19.04. 2020*	
9.	*Розробка програмного забезпечення*	*02.05. 2020*	
10.	*Налагодження програми*	*06.05. 2020*	
11.	*Виконання графічних документів*	*09.05.2020*	
12.	*Оформлення пояснювальної записки*	*22.05.2020*	
13.	*Подання ДП на попередній захист*	*29.05.2020*	
14.	*Подання ДП рецензенту*	*06.06.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Максим МАМУТА
(підпис)

Керівник _____ Юрій ОЛІЙНИК
(підпис)

Пояснювальна записка
до дипломного проєкту

на тему: Розподілене сховище даних на основі Apache Spark

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проект	2	
2	А4	КПІ.ІП-6318. 045430.02.81	Розподілене сховище даних на основі Apache Spark		
			Пояснювальна записка	71	
3	А4	КПІ.ІП-6318. 045430.03.81	Розподілене сховище даних на основі Apache Spark		
			Технічне завдання	12	
4	А4	КПІ.ІП-6318. 045430.04.51	Розподілене сховище даних на основі Apache Spark		
			Програма та	7	
			методика тестування		

АНОТАЦІЯ

Робота містить 23 рисунки і 17 таблиць.

Кожного року зростає кількість запитів на отримання рішення проблеми збереження великого обсягу даних на проектах різноманітної складності. Робота із даними і удосконалення методів їх зберігання є доволі розповсюдженою задачею на сьогоднішній день у багатьох технологічних компанія, а також у окремих розробників.

Щодня у світі з'являється велика кількість інформації, яка зберігається і структурується у сховищах і базах даних. А коли даних стає забагато для однієї бази даних, або маємо різні кінцеві цілі роботи з даними, то переходимо до використання багатьох сховищ і баз даних одночасно із встановленням взаємодій між ними, або ж готуємо сховище, яке може змінити свою внутрішню структуру у залежності від запиту користувача.

Різноманітні види і структури сховищ та баз даних є націленими на оптимальне рішення лише вузького спектру задач, і маючи сховище що може втілювати у собі декілька різних типів і підходів до зберігання та обробки даних є перевагою. Теорія аналізу і роботи із великим обсягом даних є потужним інструментом для вирішення цілого ряду практичних і аналітичних проблем, таких як прогнозування результатів певної дії чи явища, розподіл даних за ознаками, ранжування їх за вагою і роллю.

У першому розділі були розглянуті найбільш ефективні алгоритми і методи обробки великого обсягу даних для досягнення оптимального розподілу ресурсів у системі, а також певні види сховищ що краще виконують певні задачі, а також описано функціональні та нефункціональні вимоги до програмного продукту.

Під час написання другого розділу був зроблений опис архітектури програмного забезпечення та його компонентів, за методологією IDEF0 були створені схеми для бізнес процесів програмного продукту в цілому та його окремих модулів. А також було розібрано у деталях потрібні для правильної

роботи програми функції та параметри і описані сутності ключових змінних у базі даних.

Під час написання третього розділу було проведено планування тестування програмного забезпечення сховища великих даних, що може змінювати свою внутрішню структуру у залежності від запиту користувача, що зможе виявити всі потенційні дефекти програмного забезпечення, які можуть призвести до небажаної поведінки системи в цілому та можливому погіршенню досвіду використання системи користувачами.

У четвертому розділі було описано запуск програмного забезпечення і розглянути можливості до масштабування кластеру із серверів.

ВЕЛИКІ ДАНІ, СХОВИЩЕ ВЕЛИКИХ ДАНИХ, ОПТИМАЛЬНИЙ РОЗПОДІЛ РЕСУРСІВ

ABSTRACT

The work contains 23 drawings and 17 tables.

Every year, problem of storing large amounts of data on projects of varying complexity increases the number of requests for a solution to. Working with data and improving storage methods is a fairly common task today for many technology companies, as well as individual developers.

Every day, a large amount of information appears in the world, which is stored and structured in repositories and databases. And when there is too much data for one database, or we have different end goals for working with data, we move on to using many repositories and databases while interacting between them, or prepare a repository that can change its internal structure depending on the user request.

The various types and structures of repositories and databases are aimed at optimally solving only a narrow range of tasks, and having a repository that can embody several different types and approaches to data storage and processing is an advantage. The theory of analysis and work with a large amount of data is a powerful tool for solving a number of practical and analytical problems, such as predicting the results of a particular action or phenomenon, the distribution of data by characteristics, ranking them by weight and role.

The first section discusses the most efficient algorithms and methods for processing large amounts of data, achieve optimal resource allocation in the system, as well as certain types of storage that better perform certain tasks, and describes the functional and non-functional requirements for the software product.

During the writing of the second section, was made a description of the software architecture and its components, according to the IDEF0 methodology, was created schemes for business processes of the software product as a whole and its individual modules. It also analyzed in detail the functions and parameters required for the proper operation of the program and described the essence of key variables in the database.

During the writing of the third section, testing of large data warehouse software was planned, which may change its internal structure depending on the user's request, which will be able to detect all potential software defects that could lead to undesirable system behavior and possible deterioration system users.

The fourth section describes how to run the software and consider how to scale the cluster of the servers.

BIG DATA, BIG DATA STORAGE, OPTIMAL DISTRIBUTION OF RESOURCES

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СИСТЕМА – програмне забезпечення для зміни внутрішньої структури сховища.

БД – база даних.

CSV – текстовий формат зберігання даних та тип файлів, що містить заголовок із описом та дані.

JSON – текстовий формат зберігання даних у вигляді ключ-значення.

API – програмний інтерфейс доступу.

REST – формат передачі запитів і відповідей через мережу.

SQL – формат зберігання у реляційних базах даних.

NoSQL – формат зберігання у нереляційних базах даних.

Master-node – головний сервер, що керує розподіленою системою.

Data-node – допоміжних сервер у розподіленій системі, що зберігає дані.

Pipeline – це процес перетворення даних чи їх структури.

Токен – унікальний набір символів ідентифікації поточної сесії взаємодії клієнту із сервером.

ВСТУП

Кожного року зростає кількість запитів на отримання рішення проблеми збереження великого обсягу даних на проектах різноманітної складності. Робота із даними і удосконалення методів їх зберігання є доволі розповсюдженою задачею на сьогоднішній день у багатьох технологічних компанія, а також у окремих розробників. Щодня у світі з'являється велика кількість інформації, яка зберігається і структурується у сховищах і базах даних. А коли даних стає забагато для однієї бази даних, або маємо різні кінцеві цілі роботи з даними, то переходимо до використання багатьох сховищ і баз даних одночасно із організацією взаємодій між ними, або ж готуємо сховище, яке може змінити свою внутрішню структуру у залежності від запиту користувача.

Різноманітні види і структури сховищ та баз даних є націленими на оптимальне рішення лише вузького спектру задач, і маючи сховище що може втілювати у собі декілька різних типів і підходів до зберігання та обробки даних є перевагою. Теорія аналізу і роботи із великим обсягом даних є потужним інструментом для вирішення цілого ряду практичних і аналітичних проблем, таких як прогнозування результатів певної дії чи явища, розподілу даних за ознаками, ранжування їх за вагою і роллю.

Актуальність теми – дані використовуються у всіх сферах життя та побуту у різному виді, із різними якостями та відображеннями. І вирішення задачі збереження великого обсягу даних і оптимального розподілу ресурсів системи залишається актуальною і по сьогоднішній день. Сховище, яке може змінити свою внутрішню структуру в залежності від запиту користувача є більш оптимальним варіантом, який об'єднує у собі функціональні можливості і при цьому залишається економічно вигідним варіантом. Програмне забезпечення дозволяє досягати поставлені цілі та полегшити обробку великих обсягів даних.

Мета – збільшення можливості сховищ даних за рахунок організації внутрішньої структури у залежності від запитів користувача.

Призначення – суміщення декількох типів сховищ великих даних у одному, що є адаптивним до задачі користувача, для оптимізації зберігання і обробки великих даних.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Великі дані - це розділ, що розглядає способи для аналізу та обробки наборів інформації, які є занадто складними або великими, щоб їх можливо було обробляти традиційними методами та програмним забезпеченням.

Задачі, які необхідно вирішувати при роботі із великими даними включають в себе: збір даних із різноманітних джерел інформації, зберігання, обмін даними із зовнішніми джерелами та сховищами, інтелектуальний аналіз, пошук необхідної інформації в оброблених даних, візуалізація отриманих результатів обробки.

Сховища для зберігання необроблених даних можемо використовувати як відправну точку перед обробкою та аналізом даних, брати із цього сховища оригінальні та ще не видозмінені дані, які можуть мати потрібну інформацію у різних варіаціях при різних типах обробки.

1.2 Змістовний опис і аналіз предметної області

1.2.1 Типи сховищ даних

Технології SQL і по сьогодні можуть бути використані при обробці великих даних, що спрощують перехід від традиційного типу зберігання та обробки даних без значних змін кодової бази. NoSQL технології впроваджують гнучкі моделі даних, які можуть не мати певної схеми та структури, при цьому мати горизонтальну масштабованість. Ці бази даних спрямовані на полегшення управління великомасштабними проектами.

Також окремо можна виділити типи сховищ, які не використовують певної структури даних, а використовують розподілену файлову систему одного чи декількох серверів і містять у собі файли із різними типами та форматами зберігання даних.

1.2.2 Алгоритм MapReduce

MapReduce виконує перетворення, щоб розділити дані, які необхідно обробити, на невеликі частини та призначити їх декільком системам на виконання. У технічному плані алгоритм MapReduce допомагає в надсиланні завдань Map та Reduce на відповідні сервери в кластері.

MapReduce використовується для обробки паралельних завдань у великих наборах даних за допомогою великої кількості комп'ютерів (вузлів), і якщо всі вузли в одній локальній мережі та використовують аналогічне обладнання то вони разом називаються кластером, або якщо вузли поділяються між собою географічно та адміністративно та є розподіленими системами, та використовують більш неоднорідне обладнання, то вони називаються сіткою. MapReduce може скористатися локальністю даних, обробляючи їх поблизу місця, де вони зберігаються, щоб мінімізувати накладні зв’язки.

Алгоритм MapReduce містить два важливі завдання, а саме: Map і Reduce. Завдання Map виконується за допомогою класу Mapper. Завдання Reduce

виконується за допомогою класу Reducer. Клас Mapper приймає на вхід дані, видає їм індекси - токени, фільтрує, співвідносить їх до пар та сортує дані. Вихідні дані класу Mapper використовуються як вхідні дані класу Reducer, який, в свою чергу, шукає відповідні пари і зменшує їх кількість, обчислює вихідний результат групування цих пар. Обробка може відбуватися на даних, що зберігаються або у файловій системі (неструктуровані), або в базі даних (структуровані).

MapReduce працює з даними, які є структурованими у пари ключ та значення. Map приймає одну пару даних із типом в одній конфігурації даних та повертає список пар у іншій конфігурації:

Map(k1,v1) → list(k2,v2)

Потім Reduce застосовується до кожної групи пар, сформованих Map, що в свою чергу створює колекції даних у цій самій конфігурації:

Reduce(k2, list (v2)) → list((k3, v3))

1.2.3 Алгоритм Parallel PFP

Parallel Frequent Pattern це корисний інструмент для виявлення елементів, що часто зустрічаються у вибірці. Було розроблено ряд значущих алгоритмів FIM, щоб прискорити показники знаходження схожих елементів у вибірці великих даних. На жаль, коли маємо занадто великий розмір набору даних, і маємо використовувати великий обсяг пам'яті, все одно обчислювальна вартість може бути надмірно дорогою.

Для цього, почали використовувати FP-Growth algorithm, так званий паралельний алгоритм PFP на кластері розподілених машин. PFP розподіляє задачі на обробку таким чином, що кожна машина виконує самостійну групу завдань із знаходження та обробки даних. Такий розподіл ліквідує обчислювальні залежності між машинами і тим самим зв'язок між ними, прискорює пошук і обробку даних.

Враховуючи величезний список транзакцій, алгоритм знаходить усі унікальні функції (набори значень поля) та виключає ті функції, частота яких у всьому наборі даних менша. Використовуючи ці функції, що залишилися N, знаходимо шаблони K для кожного з них, генеруючи загальну кількість шаблонів NxK. Алгоритм PFP - це загальна реалізація, можемо використовувати будь-який тип об'єкта для позначення функції.

1.2.4 Алгоритм HPPC

HPPC реалізує типові колекції (list, queue, dequeue, map) зі спеціалізованими версіями, які зберігають примітивні типи без пакування їх як об'єктів. Це призводить до кращого використання пам'яті та підвищення продуктивності.

Існує декілька цілей High Performance Primitive Collections: це використовувати типові класи колекцій, які зберігають пам’ять для примітивних типів і уникають автоматичного пакування, та ціль, що швидкість кожної операції є пріоритетною.

1.3 Аналіз успішних IT- проектів

1.3.1 Аналіз відомих технічних рішень

Провівши аналіз літературних джерел що є пов’язаними із даною предметною областю, можемо стверджувати що програмне забезпечення, що працює із великими даними може бути як проектами із відкритим кодом, чи комерційними проектами із закритим кодом, та більшість із них є сервісами, що працюють у хмарі та мають веб інтерфейс для створення запитів і роботи із ними. Наш програмний продукт має можливість розвернути не тільки на хмарі, а і на своєму сервері, що у сучасних реаліях при рівномірному завантаженні є більш вигідним варіантом. Також наш проект підтримує алгоритми для більш оптимальної роботи з даними, що може видозмінювати свою внутрішню структуру за запитом користувача.

1.3.2 Аналіз відомих програмних продуктів

Звернемо свою увагу на найбільш відомі програмні продукти:

Airflow

Airflow[11] це платформа, створена спільнотою для програмного створення акторів, задля планування та моніторингу робочих процесів. Цей програмний продукт для створення, запуску і моніторингу потокових задач. Airflow має модульну архітектуру і використовує чергу повідомлень, щоб організувати довільну кількість виконуючих модулів. Пайплайни у Airflow конфігуруються кодом, що дозволяє створювати їх динамічно.

Airflow забезпечує безліч готових інтеграцій із сервісами, які готові впоратися із завданням користувача і працюють на платформі Google Cloud, веб-сервісах Amazon, Microsoft Azure та багатьох інших службах. Це робить Airflow простим у використанні у поточній інфраструктурі. Будь яка людина із знанням мови програмування Python може спробувати створити власний Airflow пайплайн.

Google Big Table

Bigtable[12] - це високоефективна система зберігання даних, побудована у файловій системі Google. Є основою Cloud Datastore, який доступний як частина хмарної платформи Google. Воно зіставляє ключ рядка та ключ стовпця із часовою позначкою і збирає все в асоціативний масив байтів. Це не реляційна база даних, а скоріше може бути охарактеризоване як розподілений, багатовимірний і сортований map. Кожна комірка великої таблиці може мати нульову або більше часових версій даних.

Bigtable призначений для масштабування в петабайтовому діапазоні на сотні чи тисячі машин, і щоб спростити додавання більшої кількості машин до системи і автоматично почати користуватися цими ресурсами без перенастроювання. Наприклад, копія Інтернету від Google може зберігатися у великій таблиці, де ключ рядка - це URL-адреса, повернена доменом, а стовпці описують різні властивості веб-сторінки та її дані.

RethinkDB

RethinkDB[13] - це перша база даних для веб-додатків у режимі реального часу, для документів із відкритим кодом, яка ефективно підтримує складні запити. RethinkDB взаємодіє із традиційними системами зберігання даних та реалізує додані лише вкладені структури зберігання даних, що робить її більш послідовною, надійною та легко реплікуваною. RethinkDB пропонує легку масштабованість та швидку відповідь на запити клієнта в режимі реального часу.

Redis

Redis[14] - це альтернативний кеш із відкритим кодом та сховище типу ключ-значення для великих даних, що забезпечує ефективну структуру даних для індексації та прискорення операцій. Redis має значні можливості для розширення в середовищі master-slave. Однак надання підтримки декільком структурам даних робить його переважним вибором для ситуацій із більш частими запитами доступу до даних.

Більш детальне порівняння із описаними рішеннями наведено у таблиці нижче:

Таблиця 1.1 – Порівняння із відомими програмними продуктами

Назва	Запуск на власному сервері	REST API	Data Querying	Вибірковий доступ до даних	Data Lake
Degree Project	+	+	+	+	+
Airflow	+	+	+	-	-
Google Big Table	-	-	+	+	+
RethinkDB	+	-	+	+	-
Redis	+	-	+	+	-

1.4 Аналіз вимог до програмного забезпечення

Під час розробки програмного забезпечення для роботи із великими даними спочатку важливо виділити ролі учасників. Клієнт, що є користувачем ПК, проводить введення, чи збір із зовнішнього джерела, великих даних і виконує один із доступних алгоритмів перетворення сховища, що є найбільш оптимальним для роботи з поточною клієнтською задачею.

Створений програмний продукт повинен містити в собі наступні функції:

– вибір способу введення великих даних;
– введення необроблених файлів із даними до HDFS;
– введення даних за допомогою CSV файлу;
– введення даних за допомогою REST API;
– вибір типу внутрішньої структури сховища;
– виконання перетворення внутрішньої структури сховища;
– аналіз результатів виконання.

Більш детально розібрана взаємодія між акторами та функціональними модулями програми для роботи із сховищем великих даних та варіанти використання програмного продукту вказано на діаграмі варіантів використання (Рисунок 1.1):

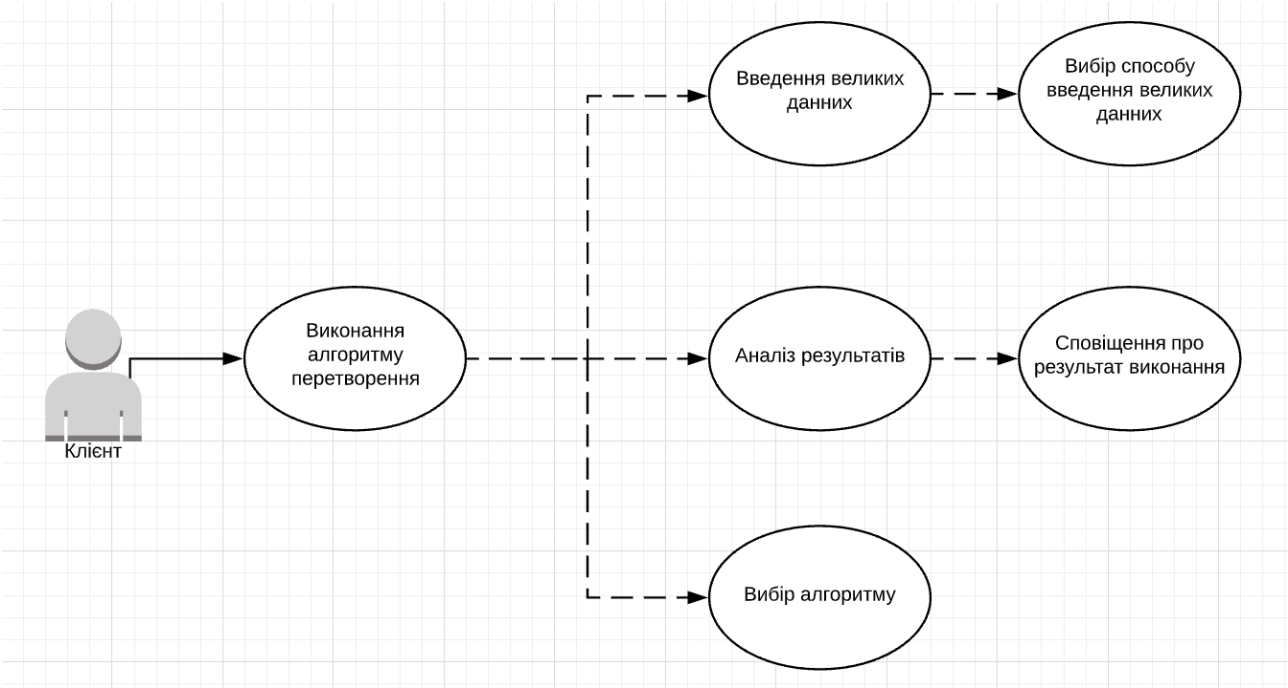


Рисунок 1.1 – Схема структурна варіантів використання

Матриця трасування вимог зображена на рисунку 1.2.

	Requirement: Введення великих даних	**Requirement:** Виконання алгоритму перетворення	**Requirement:** Аналіз результатів
Use case model: Вибір способу введення	+		
Use case model: Введення великих данних	+		
Use case model: Вибір алгоритму		+	
Use case model: Виконання алгоритму		+	
Use case model: Аналіз результатів			+
Use case model: Сповіщення про результат виконання			+

Рисунок 1.2 – Матриця трасування вимог

1.4.1 Розроблення функціональних вимог

Функціональні вимоги до програмного забезпечення для роботи зі сховищем великих даних, що може змінити свою внутрішню структуру у залежності від запиту клієнта наведені у таблиці нижче.

Таблиця 1.2 – Функціональні вимоги

Актор	Варіант використання	Функціональна вимога	Пріоритет
Клієнт	Вибір способу введення великих даних	На екрані необхідно обрати один із пунктів, що відповідають за спосіб, яким можна ввести данні	Середній
Клієнт	Введення необроблених файлів із даними до HDFS	За допомогою командної строки UNIX встановити підключення до віддаленого master-node серверу із HDFS, та використовуючи термінальні команди перемістити файли з локальної файлової системи до HDFS	Середній
Клієнт	Введення даних за допомогою csv файлу	У вікні із формою для зчитування csv файлів натиснути кнопку прикріпити файл. Після того як був відкритий каталог файлів, необхідно обрати необхідний csv файл, що містить дані	Середній
Клієнт	Введення даних за допомогою REST API	За допомогою HTTP REST програмного інтерфейсу необхідно звернутися до серверу, що має кінцеву HTTP REST точку прийому	Середній

Продовження таблиці 1.1

		даних, та передати необхідну кількість даних у форматі JSON	
Клієнт	Вибір типу внутрішньої структури сховища	На екрані необхідно обрати один із пунктів, що відповідають за спосіб, яку внутрішню структуру може мати сховище даних	Високий
Клієнт	Виконання перетворення внутрішньої структури сховища	Після вводу великих даних та вибору структури внутрішнього сховища, натиснути кнопку “Перетворення”. Після виконання алгоритмів перетворення на сторінці з’явиться блок із візуалізацією результатів виконання	Високий
Клієнт	Аналіз результатів виконання	Після виконання алгоритмів перетворення клієнт может ознайомитися із візуалізацією результатів виконання у вигляді надпису про результат та допоміжних даних при виконанні	Середній

Діаграма вимог зображена на рисунку 1.3.

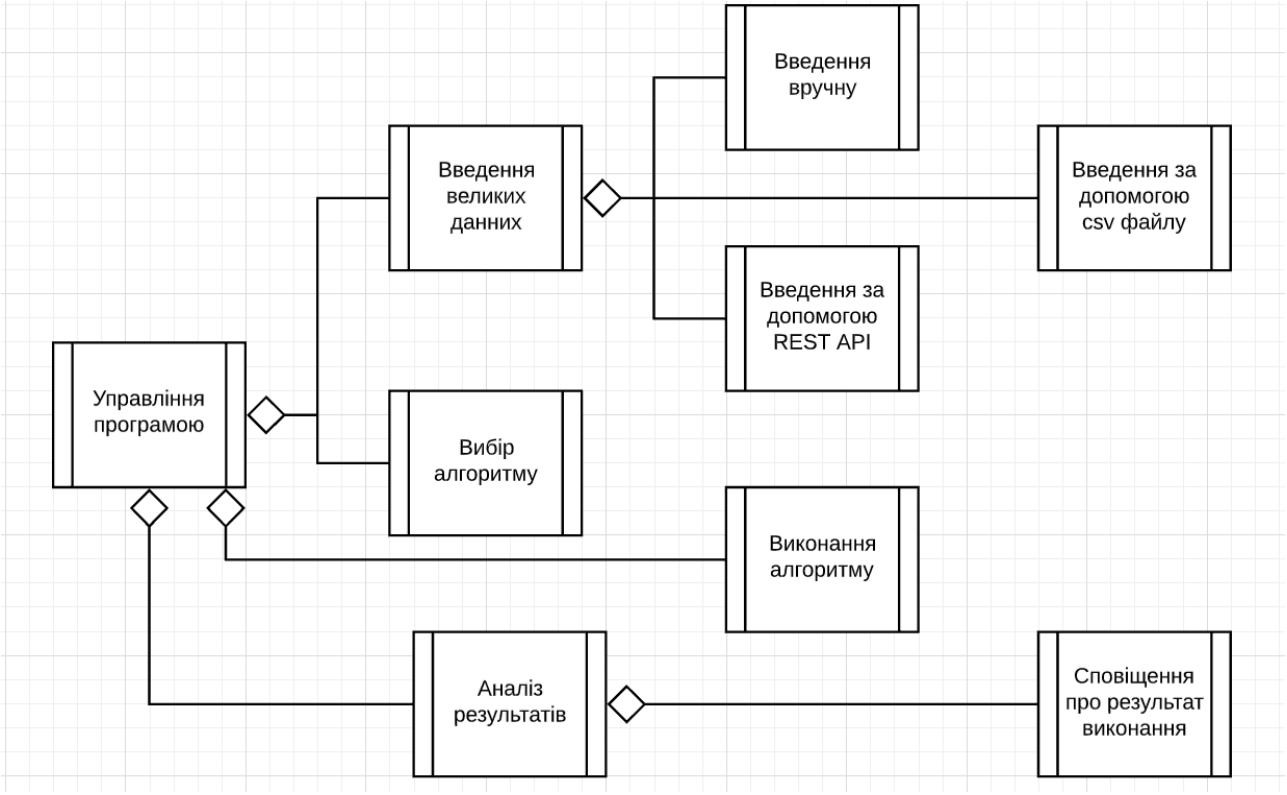


Рисунок 1.1 – Діаграма вимог

1.4.2 Розроблення нефункціональних вимог

Для використання сервісу необхідно мати один чи декілька серверів із встановленими на них операційною системою Linux одного із дистрибутивів на вибір клієнту. А також смартфон з операційною системою Android для управління всією системою. Сервери мають бути під'єднані до мережі інтернет з пропускною здатністю не менше 1 mb/s, а телефон до мережі Wi-Fi із пропускною здатністю не менше 1 mb/s.

1.4.3 Постановка завдань

Мета – розробити інтерфейс для вибору способу введення великих даних, вибору алгоритму для перетворення внутрішньої структури сховища, введення великих даних одним із доступних способів, виконання перетворення внутрішньої структури сховища за запитом, можливість отримати результати виконання та відповідні класи для реалізації функціоналу.

Призначення – виконання перетворення внутрішньої структури сховища великих даних для більш оптимальної роботи із ними.

Задачі - вибір способу введення великих даних, збір та введення великих даних, вибір алгоритму зміни внутрішньої будови сховища, виконання алгоритму перетворення, отримання результатів виконання, збереження результату виконання.

Опишемо детальніше кожну із задач.

Вибір способу введення великих даних - на екрані для введення інформації про великі дані необхідно обрати один із пунктів, що відповідають за спосіб, яким можна ввести великі данні. Якщо виберемо ручний ввід файлів із даними до HDFS, то нам потрібно буде користуватися командним рядком UNIX для переміщення файлів із локальної файлової системи до віддаленої. Якщо виберемо ввід за допомогою CSV файлу, то після натискання кнопки “Прикріпити”, можемо обрати CSV файл із локальної файлової системи та перемістити його на сервер. Якщо виберемо ввід за допомогою REST API, то система видає нам кінцеву HTTP адресу, куди можемо передавати дані за допомогою POST запитів у форматі JSON.

Збір та введення великих даних - після вибору методу введення великих даних необхідно виконати процедуру їх завантаження в систему. Після введення даних на екрані з’явиться сповіщення про результат.

Вибір алгоритму зміни внутрішньої будови сховища - після вибору методу введення великих даних та процедури їх переміщення у систему, на екрані вибору алгоритму перетворення внутрішньої структури сховища необхідно вибрати один із пунктів, що відповідає за спосіб, яким можна перетворити внутрішню будову сховища великих даних.

Виконання алгоритму перетворення - після вибору методу перетворення внутрішньої структури сховища необхідно натиснути на кнопку “Перетворити”, після чого буде запущений процес внутрішнього перетворення структури сховища для більш оптимальної роботи із великими даними в ньому.

Отримання результатів виконання - після закінчення процесу перетворення внутрішньої структури сховища можемо отримати сповіщення про результат виконання, чи був він успішний, чи трапилися помилки, які необхідно виправити і запустити процес перетворення знову.

Збереження результату виконання - після отримання успішних результатів виконання система видає інтерфейс доступу до нової структури сховища, і закриває доступ до старого. Після успішного початку взаємодії з новим інтерфейсом, система видаляє старий інтерфейс і всі допоміжні для нього файли, щоб звільнити місце на жорсткому диску віддаленої файлової системи на сервері.

1.5 Висновки по розділу

У першому розділі були розглянуті найбільш ефективні алгоритми і методи обробки великого обсягу даних для досягнення оптимального розподілу ресурсів у системі, а також певні види сховищ що краще виконують певні задачі.

Також було ознайомлено із відомими актуальними аналогами та рішеннями, описані діаграма варіантів використання, діаграма та матриця трасувань вимог, функціональні та нефункціональні вимоги до програмного продукту.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

Під час розробки якісної програми потрібно спочатку змоделювати архітектуру і внутрішні процеси програмного забезпечення. Для того, щоб спроектувати бізнес-процеси для проекту було обрано методологію розробки діаграм IDEF0. Було створено декілька рівнів бізнес процесів у проекті, схема бізнес-процесу:

- на рівні програмного продукту в цілому схематично (Рисунок 2.1);
- на рівні основних модулів програмного забезпечення схематично (Рисунок 2.2).

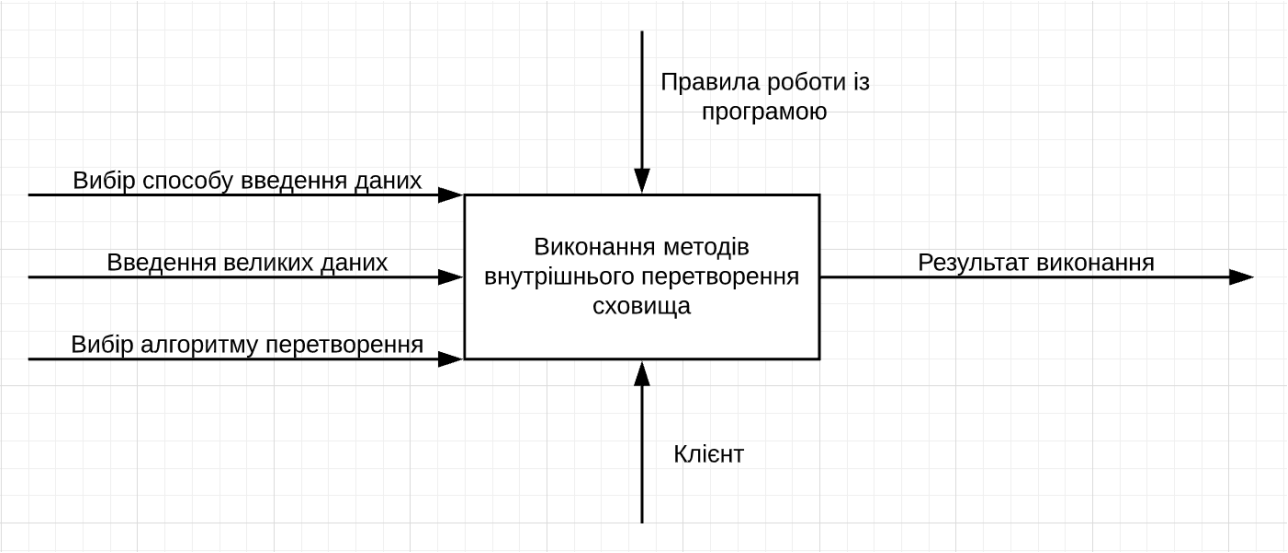


Рисунок 2.1 – IDEF0 схема на рівні програмного продукту в цілому

Рисунок 2.2 – IDEF0 на рівні основних модулів програмного забезпечення

2.2 Архітектура програмного забезпечення

Для створення серверних модулів програмного забезпечення було застосовано мультипарадигменну мову програмування Scala, та для створення клієнтських модулів програмного забезпечення було використано статично типізовану мову програмування Kotlin[15]. Оскільки нашою ціллю є створення веб-сервісу, то було доцільно використовувати Akka-фреймворк[6] для розробки серверної частини та Android SDK для розробки клієнтської частини.

Модуль Akka Http для роботи із побудований на основі архітектури MVC, яка є дуже розповсюдженою при розробці програмного забезпечення будь якого типу. Можемо представити цю архітектуру схематично (Рисунок 2.3):

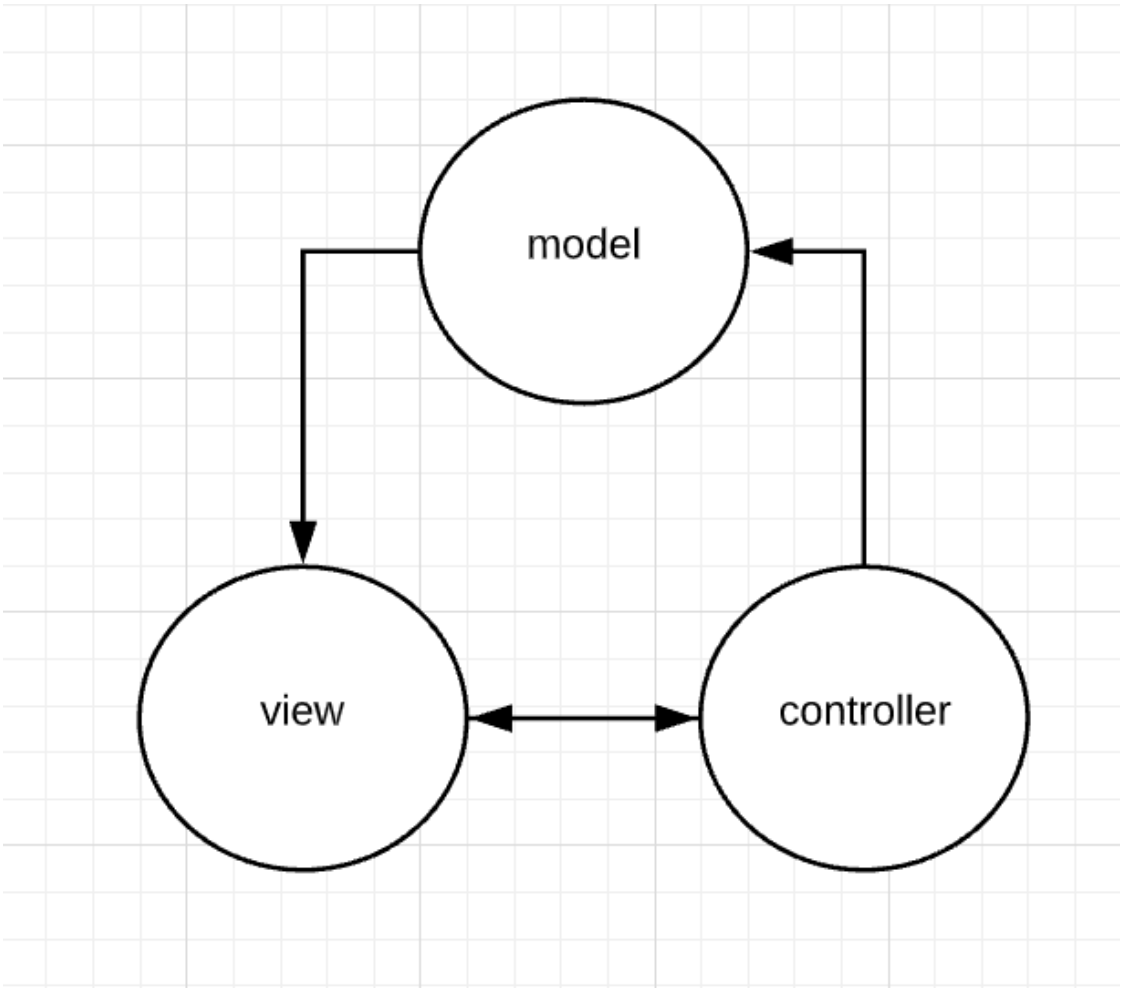


Рисунок 2.3 – Схема MVC архітектури

Опишемо основні його елементи:

- **model**: в основному має опис даних та сполучень між ними, які будуть використовуватися у програмному забезпеченні. Має в собі окремі класи, які зокрема відповідають за взаємодію із базою даних, можливими сторонніми інтеграціями із різноманітними сховищами даних чи API;

- **view**: частина програмного забезпечення, що отримує запит від користувача, обробляє отримані дані на правильність їх введення, і перенаправляє у controller для взаємодії із базою даних чи логічними видозмінами. Потім після обробки запиту і отримання із controller перенаправляє відповідь користувачу;

- **controller**: представляє собою модуль системи із програмними класами, що містять бізнес логіку додатку, та який має в собі двосторонній зв'язок із view та model, щоб отримувати запити від клієнту, обробляти їх за логікою, звертатися до бази даних при необхідності.

Модуль Akka Streams для потокової роботи із даними реалізований на основі архітектури MVVM, яка є суттєвою модифікацією стандартної та розповсюдженої архітектури MVC. Має основною ціллю обробляти запити у двосторонньому порядку від клієнта до бази даних у потоковому виді. Можемо представити цю архітектуру схематично (Рисунок 2.4):

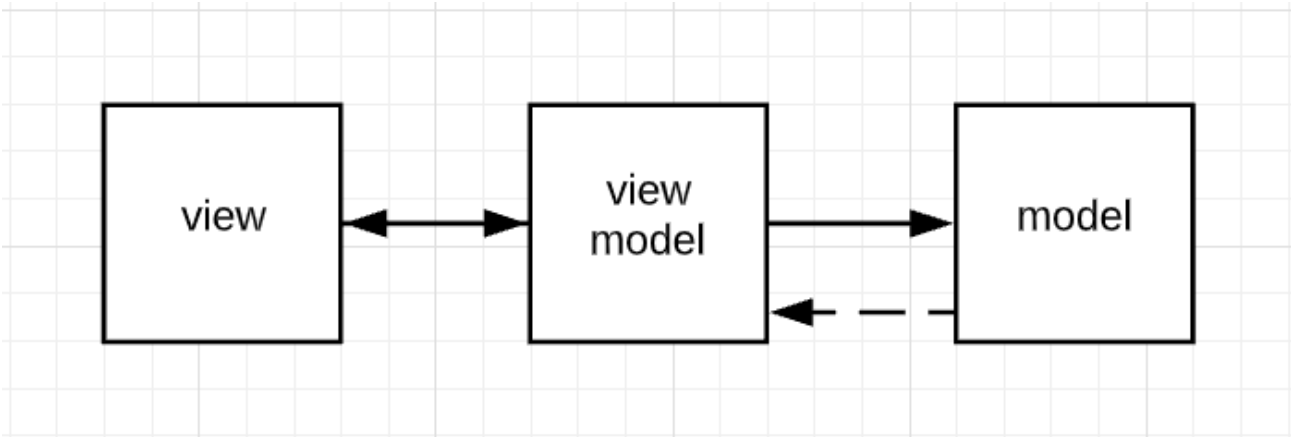


Рисунок 2.4 – Схема архітектури MVVM

Опишемо основні його елементи:

- **model**: в основному має опис даних та сполучень між ними, які будуть використовуватися у програмному забезпеченні. Має в собі окремі класи, які зокрема відповідають за взаємодію із базою даних, можливими сторонніми інтеграціями із різноманітними сховищами даних чи API;
- **view**: частина програмного забезпечення, що отримує запит від користувача, обробляє отримані дані на правильність їх введення, і перенаправляє у controller для взаємодії із базою даних чи логічними видозмінами. Потім після обробки запиту і отримання із controller перенаправляє відповідь користувачу;
- **view-model**: це абстракція подання даних, що автоматизує зв'язок між view та model у двосторонньому порядку. Тобто якщо дані змінюються на view, то вони автоматично змінюються і у model. І навпаки, якщо дані змінюються у model, то вони автоматично змінюються і на view;
- **binder**: декларативні дані та прив'язка команд в шаблоні MVVM. Дає можливість не писати логіку зв'язування view та model, якщо використовуються засоби декларування UI.

Коли клієнт із додатку робить будь який запит до API серверу, цей запит спочатку потрапляє у Url Dispatcher у Akka Http модулі сервера, а саме у view модуль REST API, де система розподіляє запити по методам controller модуля у відповідності до якого кінцевого HTTP Route посилався клієнт. Потім запит обробляється у controller і результат обробки передається назад у view, а там далі на мобільний додаток, де і відображається у графічній формі. Безпосередньо запит на перетворення внутрішньої будови сховища даних спочатку проходить через Akka Http модуль, а далі потрапляє до Akka Streams модуля, де відбувається потокове перетворення сховища даних.

В проекті було зроблено Android додаток, що отримує CSV файли з локальної файлової системи і відправляє їх на сервер, через який можна отримати інструкції як застосовувати термінальний ввід даних у сховище, чи

через REST API. Також у додатку можливо обрати метод перетворення сховища та отримати результати його виконання. Ще однією функцією додатку є те, що можливо отримати доступ до інтерфейсів доступу до вихідного перетвореного сховища. Для створення додатку був використаний Android SDK.

Для внутрішнього перетворення сховища були використані Apache Spark SQL та Apache Spark Streaming оператори та методи, а також Akka Streams API. Apache Spark разом із Akka це потужний інструмент для перетворення даних і дозволяє оптимально виконувати це перетворення на кластері із комп'ютерів.

Apache Spark[1] це уніфікований аналітичний фреймворк для роботи із великими даними на кластері із комп'ютерів, який має програмне API для роботи із багатьма мовами високого рівня, у тому числі із Scala. Spark досягає високої продуктивності як для batch даних, так і потокових даних, використовуючи оптимізатор запитів і механізм фізичного виконання. Spark пропонує понад 80 операторів високого рівня, які спрощують створення паралельних додатків на кластері із комп'ютерів. Apache Spark використовують для виконання розподіленої обробки і аналізу неструктурованих та слабо структурованих даних.

Кожна одиниця даних описується у системі парою ключ-значення. Саме це і є вхідними параметрами для вводу даних у систему. Якщо використовувати csv файл, то необхідно упевнитися, що цей файл відповідає стандарту та має заголовок із переліком ключів до полів із даними. Якщо використовується REST API для введення даних, то необхідно дотримуватися структури запитів, описаних у відповідній сторінці мобільного додатку.

Опишемо сутності для представлення одиниці даних у системі:

- ключ;
- значення;
- ідентифікатор, що видається системою.

Більш детальний опис сутності даних у системі наведено у таблиці 2.1.

Таблиця 2.1 – Опис сутності даних у системі

Назва поля	Опис	Тип	Обов’язковість
data_type	Тип структури зберігання даних	String	Так
session_id	Номер сесії зберігання даних	String	Так
is_save	Чи закінчено зберігання даних	Boolean	Так

Під час перетворення внутрішньої структури сховища, створюються окремі сесії обробки даних, котрі описуються типом та унікальним номером початкового формату даних, типом та унікальним номером вихідного формату даних, унікальним номером сесії обробки, та чи закінчений процес. А також ця таблиця містить можливий перелік помилок, якщо перетворення не було успішним. Більш детальна інформація наведена у таблиці 2.3.

Таблиця 2.3 – Опис сутності запису про перетворення внутрішньої структури сховища

Назва поля	Опис	Тип	Обов’язковість
start_data_type	Тип початкового формату даних	Int	Так
start_session_id	Унікальний номер сесії початкового переліку даних	String	Так
end_data_type	Тип кінцевого формату даних	Int	Так
end_session_id	Унікальний номер сесії кінцевого переліку даних	String	Так

Продовження таблиці 2.3

session_id	Унікальний номер сесії обробки даних	String	Так
is_successful	Чи успішне перетворення	Boolean	Так
errors	Список можливих помилок при перетворенні	List(String)	Ні

Вихідними даними є результат виконання внутрішнього перетворення сховища і відповідна відмітка у полі в базі даних. Опис внутрішніх полів системи і взаємодія із ними виконана використовуючи Akka Http фреймворк і PostgreSQL базу даних.

Опишемо сутності, які представляють собою результати перетворення внутрішньої структури сховища, тобто виконання алгоритму перетворення:

Результат перетворення даних із Datalake в Hbase:

- назва таблиці;
- назва column family;
- ідентифікатор переліку даних.

Результат перетворення даних із Datalake в Phoenix:

- перелік таблиць реляційного типу;
- назва column family у нереляційній побудові;
- ідентифікатор переліку даних.

Результат перетворення даних із Datalake в Parquet:

- назва директорії зберігання;
- результат виконання Spark операторів у процентному відношенні;
- ідентифікатор переліку даних.

Детальний опис сутностей, які представляють собою результати перетворення внутрішньої структури сховища, тобто виконання алгоритму перетворення наведено у таблиці 2.4 - 2.6.

Таблиця 2.4 – Результат перетворення із Datalake в Hbase

Назва поля	Опис	Тип	Обов’язковість
table_name	Назва таблиці	String	Так
column_family	Назва column family переліку даних	String	Так
data_id	Ідентифікатор переліку даних	String	Так

Таблиця 2.5 – Результат перетворення із Datalake в Phoenix

Назва поля	Опис	Тип	Обов’язковість
table_names	Перелік таблиць реляційного типу	List(String)	Так
column_family	Назва column family у нереляційній побудові	String	Так
data_id	Ідентифікатор переліку даних	String	Так

Таблиця 2.6 – Результат перетворення із Datalake в Parquet

Назва поля	Опис	Тип	Обов’язковість
directory_name	Назва директорії зберігання	String	Так
percent_success	Результат виконання spark операторів у процентному відношенні	Float	Так
data_id	Ідентифікатор переліку даних	String	Так

2.3 Конструювання програмного забезпечення

Опишемо методи та функції, що є складовою виконання бізнес логіки у проекті, представлені у таблиці 2.7:

Таблиця 2.7 – Специфікації функцій

Назва функції	Вхідні параметри	Вихідні параметри	Призначення
loadCSVFile	CSV файл	Результат завантаження на сервер	Завантажити CSV файл із локального сховища

Продовження таблиці 2.7

openDataSession	-	sessionId	Отримати id, що буде використаний для прив’язки даних до певного JSON файлу на сервері
putDataBySession	sessionId	Результат завантаження на сервер	Додати перелік даних до JSON файлу, прив’язаного до цієї сесії
closeDataSession	sessionId	Результат закриття сесії	закрити JSON файл, прив’язаний до конкретної сесії
putCSVToDatalake	CSV файл	-	Перемістити завантажений csv файл із серверного сховища до HDFS
putJSONToDatalake	JSON файл	-	Перемістити завантажений JSON файл із серверного сховища до HDFS

Продовження таблиці 2.7

convertFilesToHbase	Файл із Datalake	Результат конвертування	Конвертувати дані файлу і перемістити їх до Hbase бази даних
convertCSVToHbase	CSV файл	Результат конвертування	Конвертувати дані файлу і перемістити їх до Hbase бази даних
convertJSONToHbase	JSON файл	Результат конвертування	Конвертувати дані файлу і перемістити їх до Hbase бази даних
convertFilesToPhoenix	Файл із Datalake	Результат конвертування	Конвертувати дані файлу і перемістити їх до Phoenix бази даних
convertCSVToPhoenix	CSV файл	Результат конвертування	Конвертувати дані файлу і перемістити їх до Phoenix бази даних
convertJSONToPhoenix	JSON файл	Результат конвертування	Конвертувати дані файлу і перемістити їх до Phoenix бази даних

Продовження таблиці 2.7

convertFilesToParquet	Файл із Datalake	Результат конвертування	Конвертувати дані файлу і записати їх до HDFS директорії із Parquet файлами
convertCSVToParquet	CSV файл	Результат конвертування	Конвертувати дані файлу і записати їх до HDFS директорії із Parquet файлами
convertJSONToParquet	JSON файл	Результат конвертування	Конвертувати дані файлу і записати їх до HDFS директорії із Parquet файлами
exportToJSONFile	Дані зі сховища	JSON файл	Витягнути дані зі сховища та експортувати у JSON файл
exportToCSVFile	Дані зі сховища	CSV файл	Витягнути дані зі сховища та експортувати у JSON файл

2.4 Аналіз безпеки даних

При створенні сучасного програмного забезпечення необхідно забезпечити захист персональної інформації користувача. Необхідно на кожному етапі роботи програми забезпечити надійний захист від стороннього втручання.

Опишемо основні етапи роботи програми:

- введення даних у додатку;
- обробка даних у додатку;
- передача даних і запитів;
- обробка даних і запитів на сервері.

На етапі введення даних за захист даних дбає сама операційна система мобільного додатку. Завдяки тому, що кожний додаток у операційній системі Android запускається на віртуальній машині у окремому процесі, система зводить можливості втрати персональний даних майже повністю.

Обробка даних у додатку це перевірка правильності введення даних і можливості створення такого типу запиту. На цьому етапі додаток використовує систему токенів разом із сервером, щоб ідентифікувати себе і мати право зробити запит до серверу. Така система гарно зарекомендувала себе у багатьох проектах, це на сам перед система авторизації, після якої додаток отримує від серверу унікальний токен, який є правильним лише на теперішню сесію роботи додатка із сервером і представляє собою випадково згенеровано послідовність символів, яку майже неможливо швидко підібрати.

Якщо готуємо передавати користувацькі паролі на сервер, то обов'язково беремо хеш від них із сіллю, і згенеровану послідовність можемо передавати далі. Якщо необхідно передати послідовність даних, які мають бути відтворені у точно такому же вигляді, то використовуємо двостороннє шифрування даних на додатку і на сервері за допомогою криптографічної бібліотеки libsodium. Коли сервер чи додаток отримують шифрований порядок даних, разом із сіллю,

вони розшифровують це завдяки ключам, які були узгоджені ними перед початком передачі даних.

На етапі передачі даних і запитів на сервер, додаток встановлює HTTP Secure з’єднання із сервером і передає дані по зашифрованому каналу зв’язку. Коли необхідно передати дані із серверу на додаток, то увесь вище описаний процес повторюється тільки у зворотному порядку.

2.5 Висновки по розділу

Під час написання другого розділу розроблено архітектуру програмного забезпечення та його компонентів, за методологією IDEF0 були створені схеми для бізнес процесів програмного продукту в цілому та його окремих модулів.

Також було розібрано у деталях потрібні для правильної роботи програми функції та параметри і описані сутності ключових змінних у базі даних.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вступ

Для того, щоб визначити ступінь готовності програмного продукту та визначення ступеня відповідальності реальної поведінки до очікуваної поведінці, потрібно розробити план тестування програмного забезпечення. План тестування, що розглянемо у цьому розділі буде базуватися на міжнародному стандарті IEEE 829-2008. Тестування буде відбуватися на рівні “системного тестування” програмного забезпечення. Метою даного тестування є:

- зробити перелік інструментів, що будуть використані у процесі тестування;
- визначити спосіб проведення тестів;
- зробити перелік функцій, що будуть підлягати тестуванню та підготувати вимоги до тестового середовища;
- забезпечити високий рівень захисту та безпеки даних користувача;
- зробити перевірку відповідності дизайну до вимог описаного технічного завдання;
- перевірити працездатність користувацького інтерфейсу додатку;
- зробити перевірку коректності результатів, що отримані під час виконання алгоритмів перетворення.

Будуть використані наступні методи тестування: функціональне, тестування системної продуктивності, тестування користувацького інтерфейсу та системи в цілому. Тестування буде виконуватися завдяки інструментам Unit test та Scala test.

Працездатність всієї системи буде перевірятися завдяки:

- динамічне ручне тестування - спроби введення спочатку коректних, а потім некоректних даних;

- тестування на відповідність основним функціональним вимогам до програмного забезпечення;
- методи статичного тестування програмного коду застосунку та серверу;
- тестування користувацького інтерфейсу додатку.

3.2 Об’єкти тестування

Програмний продукт складається із клієнтської та серверної частини, обидві із яких мають бути протестованими.

Клієнтська частина має бути протестована у відповідності до умов:

- операційна система Android версії 5.0 та вище (API 21+);
- об’єм операційної пам’яті не менше 256 Mb;
- об’єм дискового простору не менше 150 Mb.

Серверна частина має бути протестована у відповідності до умов:

- операційна система Ubuntu Server версії 16.04 та вище;
- об’єм оперативної пам’яті не менше 1 Gb;
- об’єм дискового простору не менше 30 Gb;
- кількість ядр центрального процесору не менше 2.

3.3 Функціональність, що підлягає тестуванню

Під час тестування має бути перевірена уся функціональна складова програмного забезпечення. Функції програмного забезпечення, що підлягають тестуванню представлені у відповідних таблицях:

- вибір методу введення великих даних (таблиця 3.1);
- додавання CSV файлу із локальної файлової системи до віддаленої (таблиця 3.2);
- вибір алгоритму перетворення внутрішньої будови сховища (таблиця 3.3);
- демонстрація результатів перетворення чи помилки (таблиця 3.4);

- демонстрація інтерфейсів доступу до перетвореного сховища (таблиця 3.5);
- завантаження CSV файлу із перетвореними даними (таблиця 3.6);
- завантаження JSON файлу із перетвореними даними (таблиця 3.7).

Таблиця 3.1 – Вибір методу введення великих даних

Мета тесту	Перевірити можливість для вибору способу введення великих даних
Початковий стан	Відкрита відповідна сторінка мобільного додатку із списком методів введення великих даних
Вхідні дані	Спосіб введення великих даних
Схема проведення тесту	Необхідно натиснути на відповідний пункт меню вибору методу введення великих даних
Очікуваний результат	Відкритий відповідний екран додатку до пункту меню вибору методу введення великих даних, який був обраний
Стан програмного забезпечення після закінчення тестування	Відкритий відповідний екран додатку до обраного пункту меню вибору методу введення великих даних

Таблиця 3.2 – Додавання CSV файлу із локальної файлової системи до віддаленої

Мета тесту	Перевірити можливість додавання даних до сховища за допомогою CSV файлу
Початковий стан	Відкрита сторінка мобільного додатку із методом введення даних за допомогою CSV файлу
Вхідні дані	CSV файл, що містить дані
Схема проведення тесту	Необхідно натиснути на кнопку “прикріпити” на екрані додатку та обрати бажаний CSV файл із списку внутрішніх файлів у системі

Продовження таблиці 3.2

Очікуваний результат	Розпочнеться процес завантаження файла на сервер і на екрані відобразиться колесо очікування
Стан програмного забезпечення після закінчення тестування	Колесо очікування зникне та відобразиться результат завантаження файлу на сервер чи помилка

Таблиця 3.3 – Вибір алгоритму перетворення внутрішньої будови сховища

Мета тесту	Перевірити можливість для вибору алгоритму внутрішнього перетворення сховища даних
Початковий стан	Відкрита відповідна сторінка мобільного додатку із списком алгоритмів перетворення
Вхідні дані	Алгоритм перетворення сховища та перелік даних, які необхідно перетворити
Схема проведення тесту	Необхідно натиснути на відповідний пункт меню вибору алгоритму перетворення внутрішньої структури сховища даних, вибрати один із пунктів випадаючого меню, звідки брати дані та натиснути на кнопку перетворення
Очікуваний результат	Розпочнеться процес перетворення внутрішньої структури сховища даних і на екрані відобразиться колесо очікування
Стан програмного забезпечення після закінчення тестування	Колесо очікування зникне та відобразиться результат перетворення внутрішньої структури сховища даних чи помилка

Таблиця 3.4 – Демонстрація результатів перетворення чи помилки

Мета тесту	Перевірити функціональність додатку для відображення результатів перетворення внутрішньої структури сховища чи помилки
Початковий стан	Відкрита відповідна сторінка мобільного додатку із списком алгоритмів перетворення, обраний один із алгоритмів та звідки брати дані, натиснута кнопка перетворення та процес очікування завершився
Вхідні дані	Алгоритм перетворення сховища та перелік даних, які необхідно перетворити
Схема проведення тесту	Необхідно натиснути на відповідний пункт меню вибору алгоритму перетворення внутрішньої структури сховища даних, вибрати один із пунктів меню, звідки брати дані та натиснути на кнопку перетворення, дочекатися результату
Очікуваний результат	Закінчиться процес перетворення внутрішньої структури сховища даних і на екрані зникне колесо очікування, а з’явиться текстовий результат успішного перетворення та відповідний символ “галочки”, або з’явиться екран із описом помилки виконання
Стан програмного забезпечення після закінчення тестування	Колесо очікування зникне та відобразиться результат перетворення внутрішньої структури сховища даних чи помилка

Таблиця 3.5 – Демонстрація інтерфейсів доступу до перетвореного сховища

Мета тесту	Перевірити можливість для вибору інтерфейсу доступу до сховища даних
Початковий стан	Відкрита відповідна сторінка мобільного додатку із списком інтерфейсів доступу до сховища даних
Вхідні дані	Тип інтерфейсу доступу до сховища даних
Схема проведення тесту	Необхідно перейти до відповідного екрану додатку та обрати для себе потрібний інтерфейс доступу, для використання його у сторонньому програмному забезпеченні
Очікуваний результат	Після відкриття відповідного екрану додатку побачимо список із доступних інтерфейсів доступу до сховища даних
Стан програмного забезпечення після закінчення тестування	Після відкриття відповідного екрану додатку побачимо список із доступних інтерфейсів доступу до сховища даних

Таблиця 3.6 – Завантаження CSV файлу із перетвореними даними

Мета тесту	Перевірити можливість завантаження CSV файлу із перетвореними даними
Початковий стан	Відкрита відповідна сторінка мобільного додатку із списком варіантів завантаження файлів із перетвореними даними
Вхідні дані	Тип файлу для завантаження та перетворені дані зі сховища
Схема проведення тесту	Необхідно перейти до відповідного екрану додатку та обрати для себе потрібний тип файлу, після чого натиснути на кнопку завантаження

Продовження таблиці 3.6

Очікуваний результат	Після натискання кнопки завантаження розпочнеться процес і по закінченню додаток відкриє доступний метод у системі для відображення даних завантаженого файлу
Стан програмного забезпечення після закінчення тестування	Після натискання кнопки завантаження розпочнеться процес і по закінченню додаток відкриє доступний метод у системі для відображення даних завантаженого файлу

Таблиця 3.7 – Завантаження JSON файлу із перетвореними даними

Мета тесту	Перевірити можливість завантаження JSON файлу із перетвореними даними
Початковий стан	Відкрита відповідна сторінка мобільного додатку із списком варіантів завантаження файлів із перетвореними даними
Вхідні дані	Тип файлу для завантаження та перетворені дані зі сховища
Схема проведення тесту	Необхідно перейти до відповідного екрану додатку та обрати для себе потрібний тип файлу, після чого натиснути на кнопку завантаження
Очікуваний результат	Після натискання кнопки завантаження розпочнеться процес і по закінченню додаток відкриє доступний метод у системі для відображення даних завантаженого файлу
Стан програмного забезпечення після закінчення тестування	Після натискання кнопки завантаження розпочнеться процес і по закінченню додаток відкриє доступний метод у системі для відображення даних завантаженого файлу

3.4 Функціональність, що не підлягає тестуванню

- доступність, тобто можливість використати застосунок людьми з обмеженими можливостями;
- властивість застосунку, коли користувачам дуже легко ознайомитися і почати користуватися всіма його можливостями та функціями.

3.5 Методологія проведення тестування

Маємо перевірити на достовірність правильну роботу кожного варіанту (test case), як ще кажуть, прецеденту. Результат виконання тесту та службова інформація про його перебіг буде збережена до сервісу, завдяки якому виконується тестування. Людина, що проводить процес тестування, виконує та перевіряє кожний із тестів і відмічає результат його виконання за умовами.

Після того, як тестування було завершено, буде виконано перегляд звіту із тестування. У разі знаходження будь-якої із помилок - їх перелік передається розробнику сервісу для їх подальшого виправлення.

3.6 Критерії проходження чи провалу тестування

Основна функціональність програмного забезпечення для перетворення внутрішньої структури сховища даних та всіх його складових мають функціонувати як очікувалося та бути окресленими в окремих випадках тестування. Не повинно бути виявлено критичних дефектів, тобто дефектів, при яких основна функціональність системи не виконує, або тільки частково виконує свою задачу.

Кінцевий користувач має мати змогу успішно вибрати метод введення великих даних, ввести їх, вибрати алгоритм внутрішнього перетворення сховища, виконати алгоритм внутрішнього перетворення сховища, побачити результат виконання чи помилку, побачити список доступних інтерфейсів перетвореного сховища даних, завантажити перетворені дані одним із доступних методів.

Не менше ніж 90% від усіх тестових випадків мають бути успішно пройдені. Жоден із невдало пройдених випадків тестування не повинен мати вирішального значення для можливостей кінцевого користувача використовувати систему.

3.7 Критерії призупинення тестування

Необхідно невідкладно зупинити процес тестування, якщо у системі спостерігаються дефекти у алгоритмах внутрішнього перетворення сховища.

3.8 Вимоги до тестового середовища

Тестове середовище має відповідати мінімальним показникам для коректної роботи системи, сховище даних повинно містити відповідні розділи HDFS, які мають бути протестовані, локальне файлове сховище застосунку повинно мати тестовий набір даних у файлі, який буде завантажуватися для тестування.

3.9 Відповідальність

Мати відповідальність за своєчасне дотримання розкладу виконання тестування, та за своєчасне виявлення дефектів у системі є розробник програмного забезпечення.

3.10 Розклад

Розпочатися процес тестування має за 3 тижні до дати здачі програмного забезпечення в експлуатацію.

3.11 Схвалення

Розробник може стверджувати, що тестування закінчилося успішно та визначити цей момент тільки тоді, коли створене програмне забезпечення набуває таких властивостей, що роблять його придатним до експлуатації у

промислових умовах, та коли воно набуває своїх основних функціональних особливостей.

3.12 Аналіз якості програмного забезпечення

Функціональність:

- придатність програмного забезпечення;
- захищеність програмного продукту.

Надійність:

- відмовостійкість програми;
- завершеність програми.

Зручність використання:

- зрозумілість користувацького інтерфейсу;
- можливість вивчати інтерфейс програми.

Можливість супроводжувати:

- гнучкість використання і доповнення програмного забезпечення;
- можливість тестування програми.

Таблиця 3.8 – Відношення впливу до якості

Характеристика впливу / Харак. Росту якості		Функціональність		Надійність	
		Придатність	Захищеність	Відмовостійкість	Завершеність
Функціональність	Придатність	+	-	-	+
	Захищеність	-	+	-	+
Надійність	Відмовостійкість	-	-	+	-
	Завершеність	+	-	-	+
Зручність використання	Зрозумілість	-	-	-	-
	Можливість навчатися	-	-	-	-
Супроводжуваність	Гнучкість	-	-	-	-
	Можливість тестувати	-	-	-	-

Продовження таблиці 3.8

Характеристика впливу / Харак. Росту якості		Зручність використання		Супроводжуваність	
		Зрозумілість	Навчаємість	Гнучкість	Тестованість
Функціональність	Придатність	-	-	-	-
	Захищеність	-	-	-	-
Надійність	Відмовостійкість	-	-	-	-
	Завершеність	-	-	-	-
Зручність використання	Зрозумілість	+	+	+	-
	Можливість навчатися	+	+	-	-
Супроводжуваність	Гнучкість	+	+	+	-
	Тестованість	+	+	-	-

Таблиця 3.9 - Відображення моделі зовнішньої якості на експлуатаційну якість

Характеристика впливу / Харак. / Росту якості		Експлуатаційна якість			
		Продуктивність	Задоволеність	Результативність	Безпечність
Функціональність	Придатність	+	-	-	-
	Захищеність	+	-	-	-
Надійність	Відмовостійкість	-	-	-	-
	Завершеність	+	-	-	-
Зручність використання	Зрозумілість	-	-	-	-
	Можливість навчатися	-	+	-	-

3.13 Висновки до розділу

Під час написання третього розділу було проведено планування тестування програмного забезпечення сховища великих даних, що може змінювати свою внутрішню структуру у залежності від запиту користувача, що зможе виявити всі потенційні дефекти програмного забезпечення, які можуть призвести до небажаної поведінки системи в цілому та можливому погіршенню досвіду використання системи користувачами.

Були складені таблиці, завдяки яким ми можемо планово провести тестування і виявити усі можливі дефекти та потенціальні проблеми у програмному забезпеченні.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Для того, щоб мати можливість розгорнути програмне забезпечення необхідно мати персональний комп'ютер із робочою операційною системою, бажано UNIX-сумісною. Також необхідно мати встановлену мову програмування Scala версії 2.12, JDK версії 1.8, менеджер пакетів та встановлений GIT. Спочатку нам необхідно клонувати репозиторій із програмним кодом у будь-яку папку у файловій системі ПК (Рисунок 4.1):

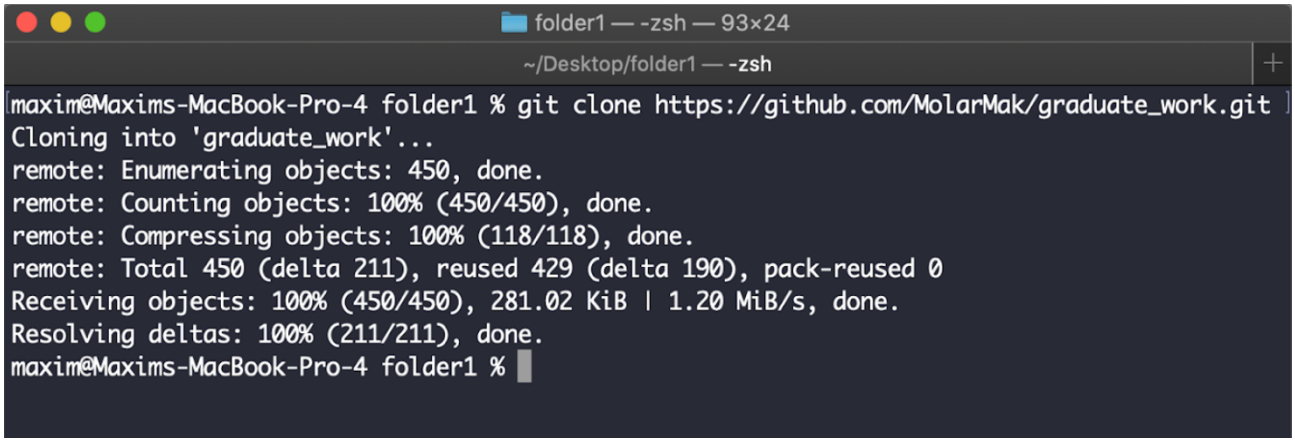


Рисунок 4.1 – Клонування Git репозиторію

Далі, для коректної роботи всієї системи нам необхідно запустити внутрішні сервіси сховища даних. По перше нам необхідно запустити розподілену файлову систему HDFS. Для цього на офіційному сайті Apache Hadoop у розділі Завантажень обираємо бажану версію HDFS і завантажуємо архівом. У проекті використовується Apache Hadoop версії 2.9.2, а тому для коретної роботи усього програмного забезпечення бажано встановити саме його. Після цього експортуємо архів із HDFS у будь-яку папку у файловій системі ПК. Далі необхідно задати конфігурації усіх активних вузлів розподілений системи. У директорії HDFS переходимо до редагування файлу etc/hadoop/core-site.xml і додаємо до нього наступні рядки (Рисунок 4.2):

```
<configuration>
   <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:9000</value>
   </property>
</configuration>
```

Рисунок 4.2 – Рядки, які необхідно додати до core-site.xml

Переходимо до редагування файлу etc/hadoop/hdfs-site.xml: і додаємо до нього наступні рядки (Рисунок 4.3):

```
<configuration>
   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>
</configuration>
```

Рисунок 4.3 – Рядки, які необхідно додати до hdfs-site.xml

Далі необхідно згенерувати ключі доступу до HDFS і необхідні команди зображені на (Рисунок 4.4):

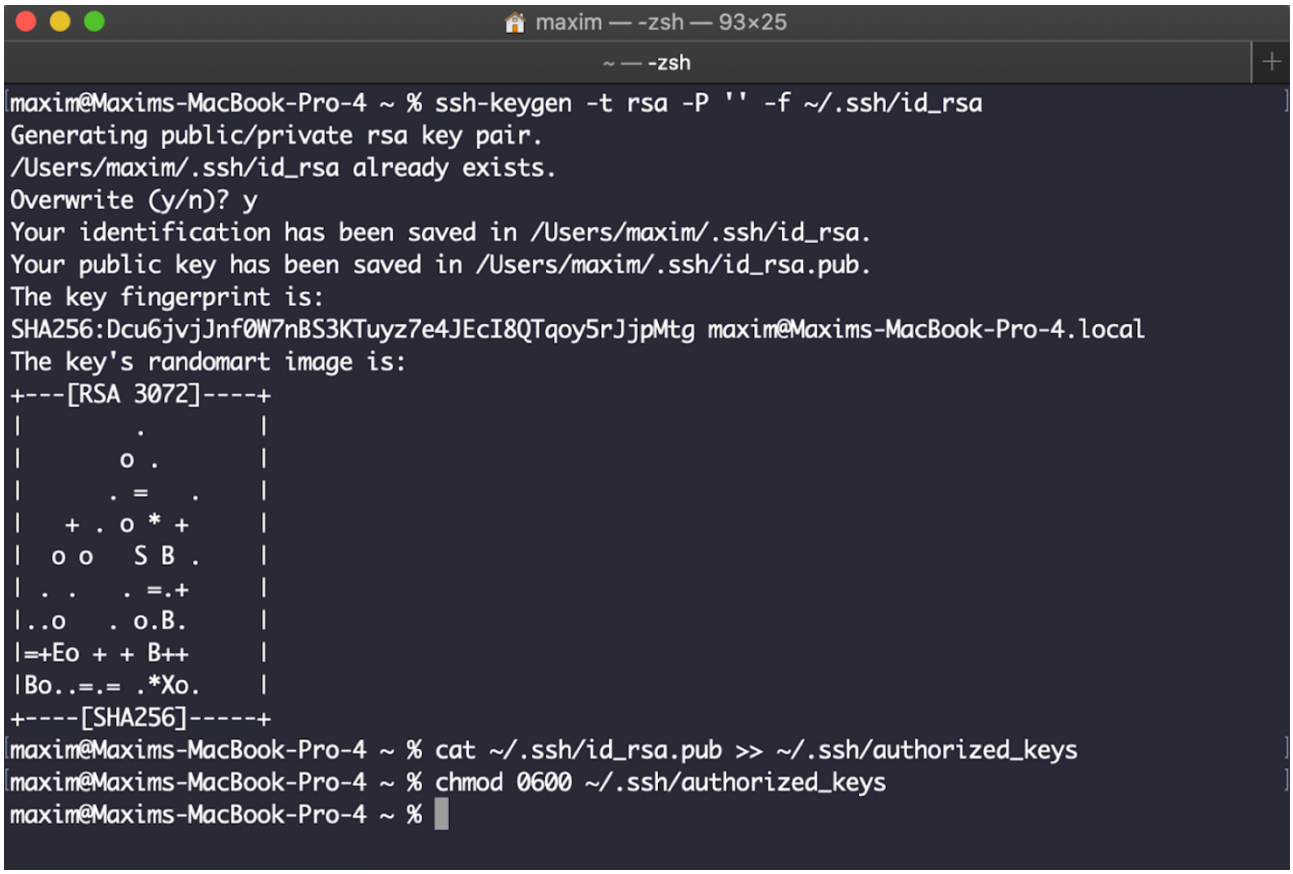


Рисунок 4.4 – Команди для генерації ключів доступу до HDFS

При першому старті HDFS нам необхідно відформувати усе сховище, і це зображено на (Рисунок 4.5):

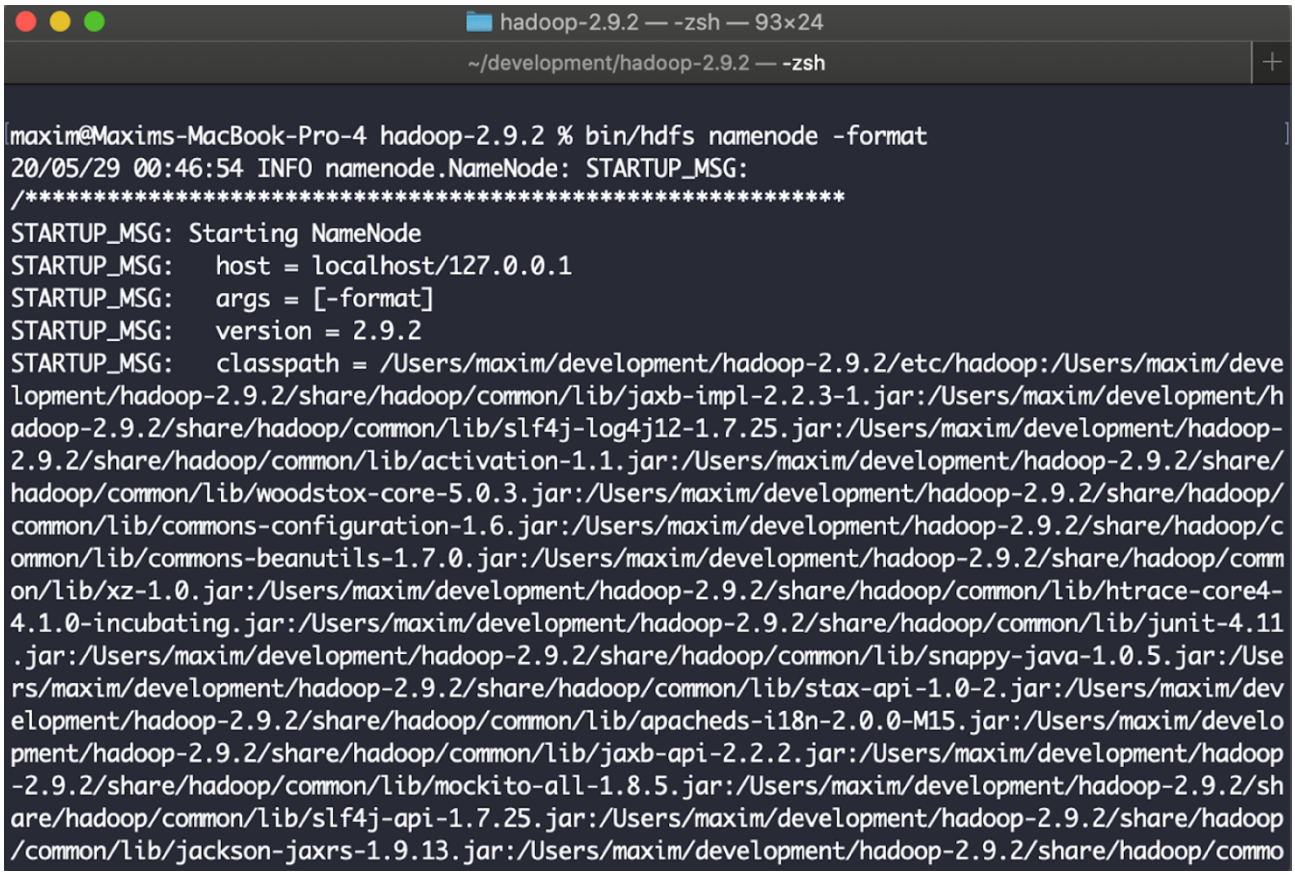


Рисунок 4.5 – Форматування HDFS сховища при першому старті

Далі можемо стартувати HDFS за допомогою команди (Рисунок 4.6), при цьому нам необхідно буде декілька разів ввести користувацький пароль від системи:

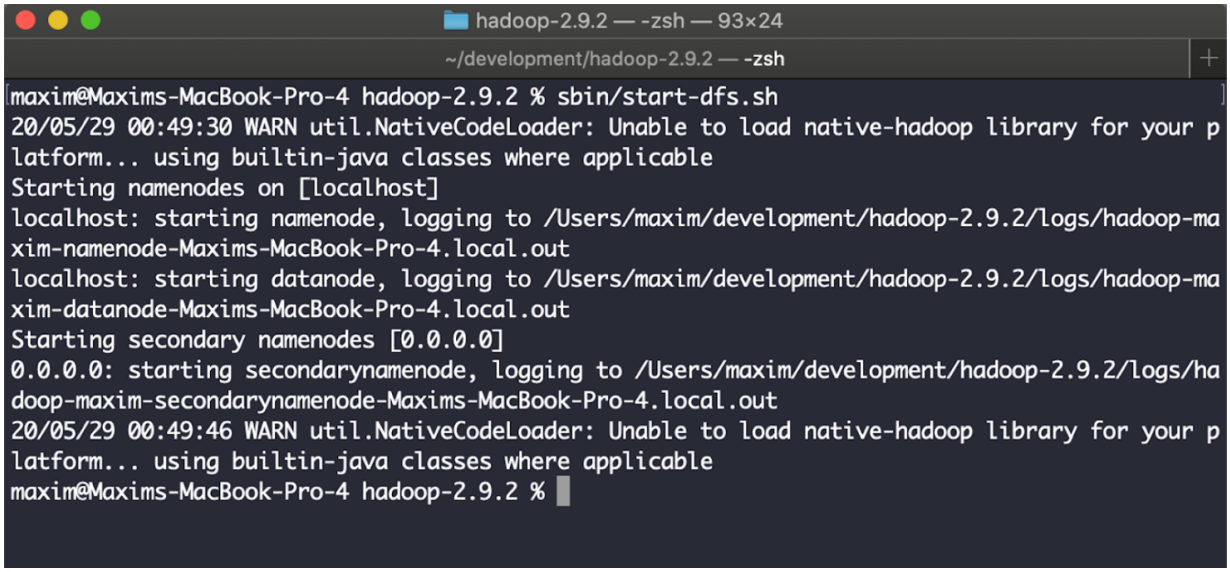


Рисунок 4.6 – Запуск HDFS

Перевіримо, що HDFS запущений, для цього перейдемо на його веб графічне відображення, яке працює на порту 50070, для цього у адресному рядку браузера введемо http://localhost:50070/ (Рисунок 4.7):

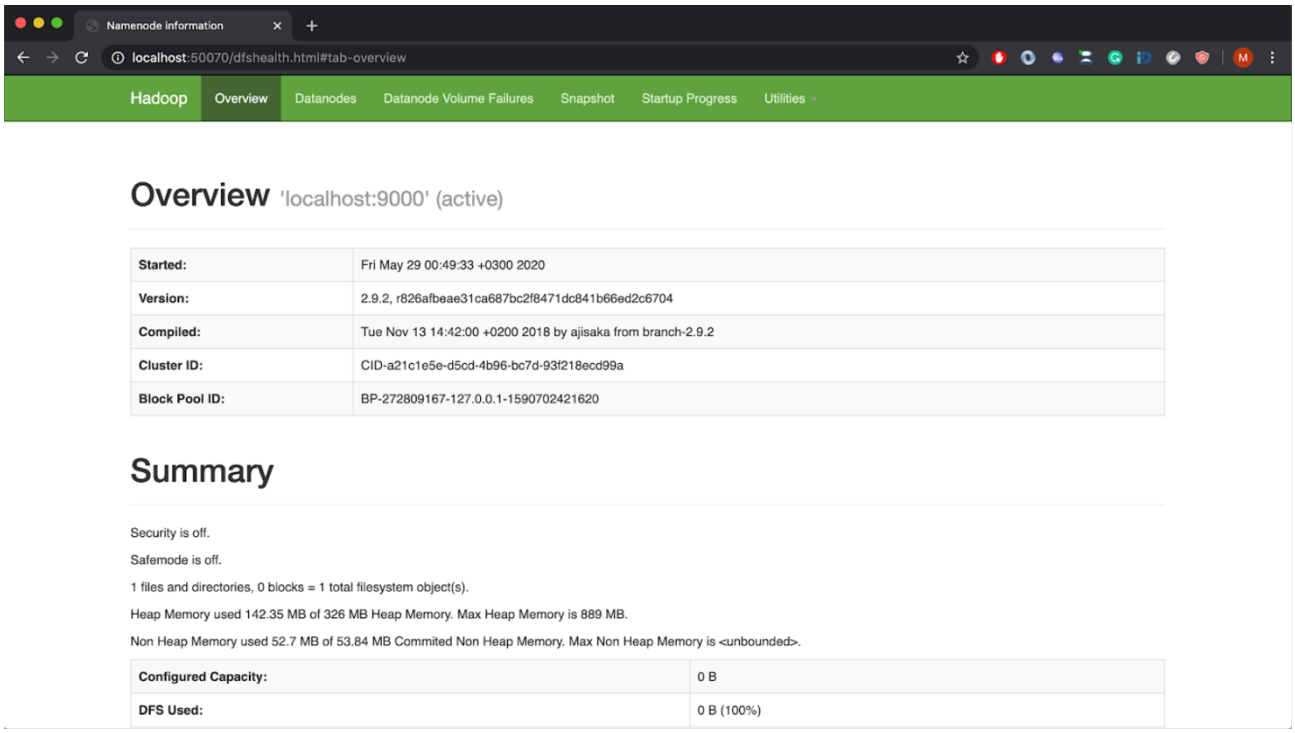


Рисунок 4.7 – Веб відображення HDFS

Далі, для коректної роботи всієї системи необхідно запустити внутрішні сервіси сховища даних, які працюють поверх HDFS. Це є сервіси Apache Hbase та Apache Hadoop. Спочатку запустимо Apache Hbase. Для цього на офіційному сайті Apache Hbase у розділі завантажень обираємо бажану версію Hbase і завантажуємо архівом. У проекті використовується Apache Hbase версії 1.4.9, а тому для коректної роботи усього програмного забезпечення бажано встановити саме його. Після цього експортуємо архів із Hbase у будь-яку папку у файловій системі ПК. Тепер можемо запустити Hbase, виконати скрипт запуску, що знаходиться у папці bin/start-hbase.sh (Рисунок 4.8):

Рисунок 4.8 – Запуск Hbase

Перевіримо коректність запуску Hbase, перейшовши до термінального режиму взаємодії із нею, для цього необхідно виконати команду bin/hbase shell (Рисунок 4.9):

Рисунок 4.9 – Перевірка запуску Hbase

Далі необхідно запустимо Apache Phoenix. Для цього на офіційному сайті Apache Phoenix у розділі завантажень обираємо бажану версію Phoenix і завантажуємо архівом. У проекті використовується Apache Phoenix версії 4.14.1, а тому для коретної роботи усього програмного забезпечення бажано встановити саме його. Після цього експортуємо архів із Phoenix у будь-яку папку у файловій системі ПК. Тепер можемо перевірити роботу Phoenix, виконавши скрипт bin/sqlline.py (Рисунок 4.10):

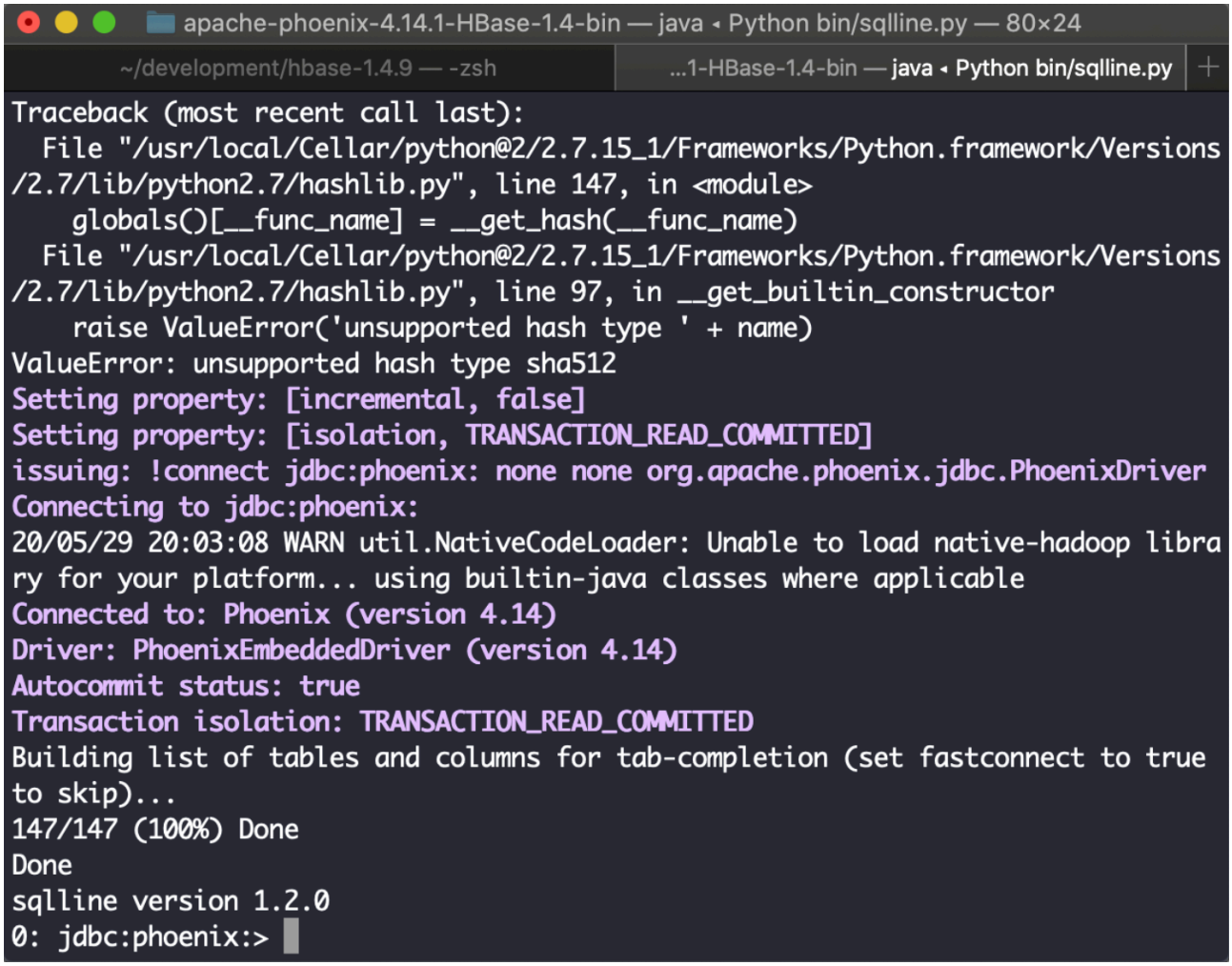


Рисунок 4.10 – Запуск Apache Phoenix

Зараз маємо повністю функціонуюче оточення для запуску нашого власного серверу, який буде працювати із даними, перетворювати їх та видавати користувачу. Виконаємо запуск серверу за допомогою команди (Рисунок 4.11):

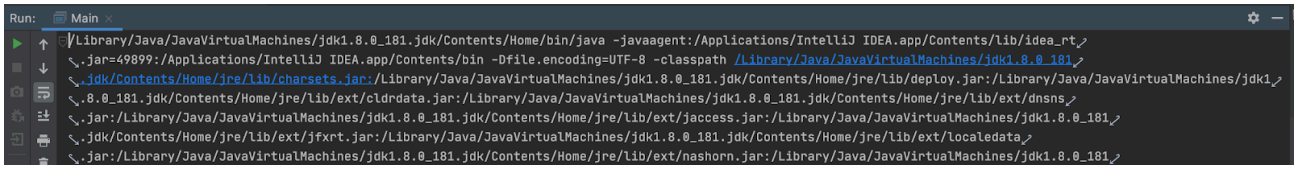


Рисунок 4.11 – Запуск серверу

4.2 Робота з програмним забезпеченням

Для того, щоб почати роботу із нашим сервісом, необхідно запустити додаток на будь-якому Android телефоні із версією ОС 5+. Після встановлення

apk додатку на телефон, запускаємо його і потрапляємо на головну сторінку додатку (Рисунок 4.12):

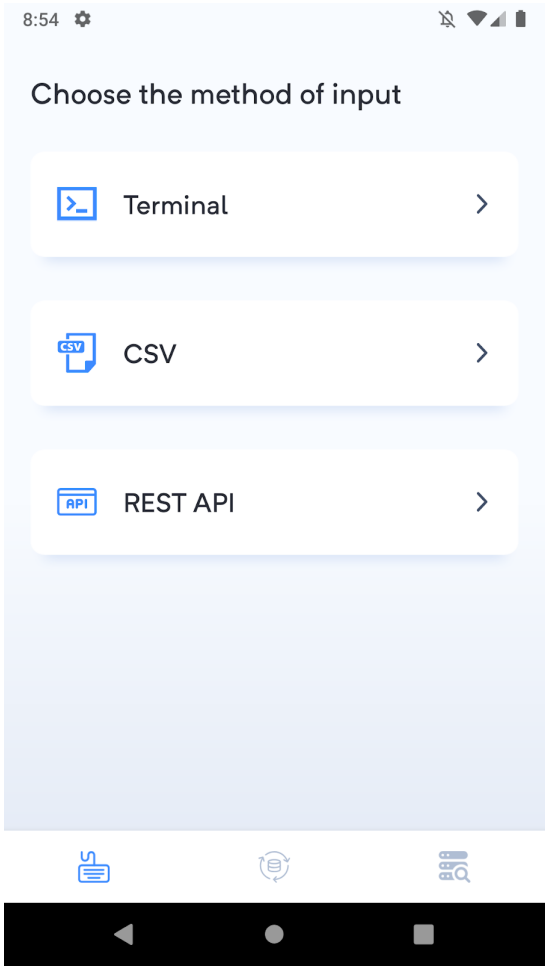


Рисунок 4.12 – Головний екран додатку

На цьому екрані можемо обрати один із доступних варіантів завантаження великих даних у систему: через термінал, CSV файл або за допомогою REST API. Якщо виберемо варіант термінального введення, то перейдемо на сторінку із детальним описом, як саме можемо ввести дані. Також якщо перейдемо на екран із методів введення через REST API, то також отримаємо детальну інструкцію, як можемо це зробити, ці два методи зображені на (Рисунок 4.13):

Рисунок 4.13 – Екрани допомоги із термінальним та REST API методами введення даних

Якщо виберемо варіант із методів введення це CSV файл, то отримаємо можливість прикріпити CSV файл із внутрішньої файлової системи мобільного телефону. Для того, щоб це виконати, необхідно натиснути на картинку “прикріпити”, обрати необхідний CSV файл у системі та очікувати його завантаження. Екран, де можемо прикріпити CSV файл і завантажити його до системи зображений на (Рисунок 4.14):

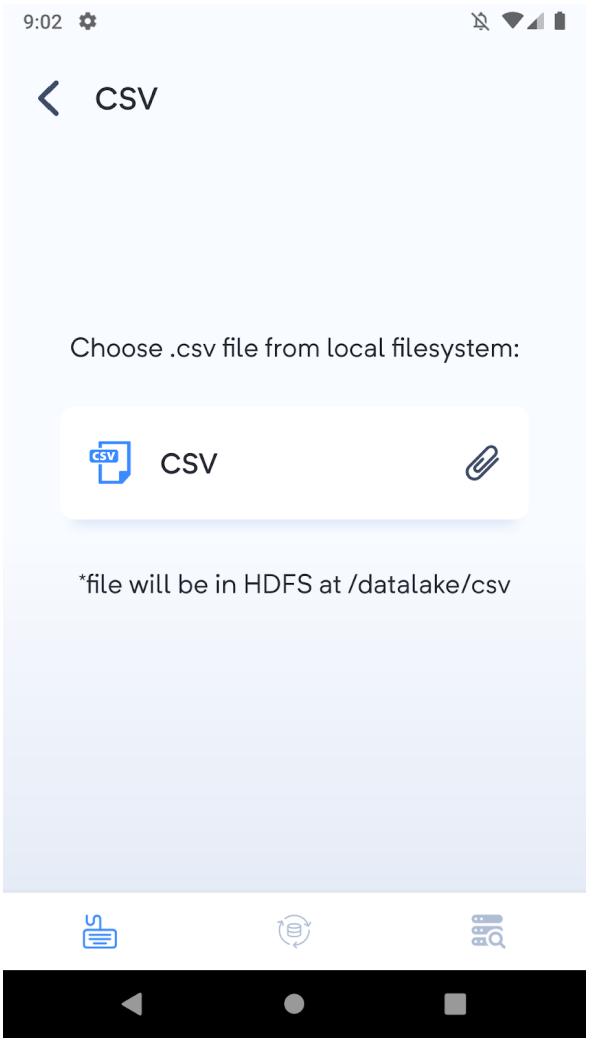


Рисунок 4.14 – Екран прикріплення CSV файлу

Далі можемо перейти до методів внутрішнього перетворення структури сховища (центральна вкладка у навігаційному меню). На цьому екрані можемо обрати на вибір один із методів внутрішнього перетворення сховища, такі як: Hbase, що є дуже ефективним для вибіркового пошуку даних у сховищі, Phoenix, який можна використовувати із SQL операціями та Parquet, який є оптимальним варіантом зберігання даних для системи, яка працює із Apache Spark. Далі можемо обрати один із можливих джерел даних у нашому сховищі, такі як: CSV, files та JSON. Далі можемо натиснути на кнопку перетворення та процес розпочнеться. Цей екран зображено на (Рисунок 4.15):

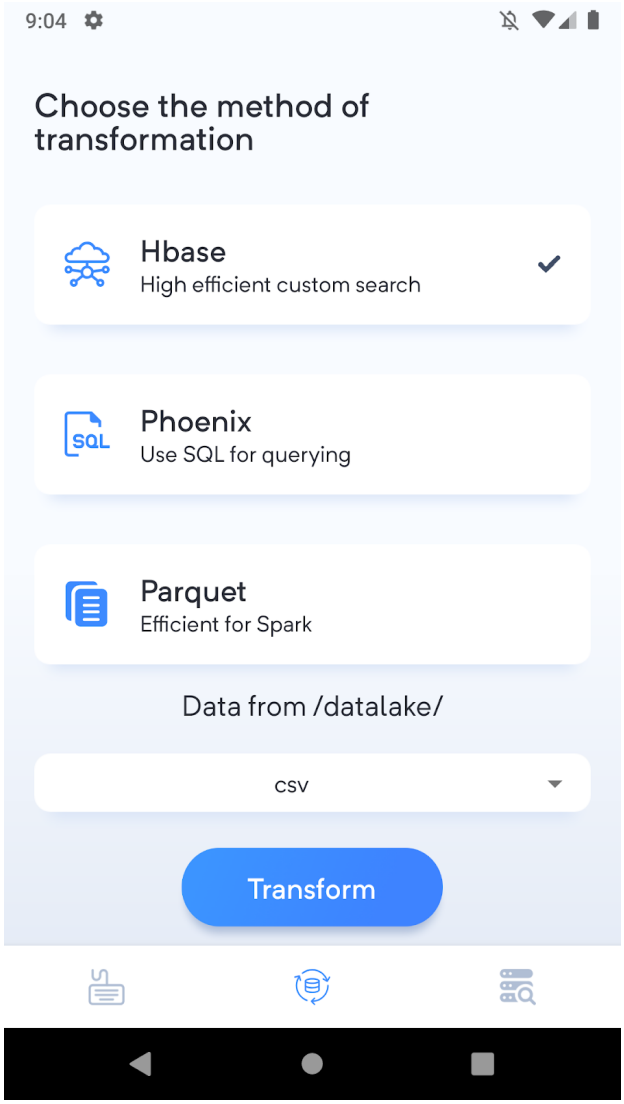


Рисунок 4.15 – Екран вибору методу перетворення

Далі перейдемо на екран, де зображені методи доступу до перетворених даних сховища, а саме інтерфейси доступу до Hbase, Phoenix чи Parquet файлів. У цих інтерфейсах зображені порт, до якого необхідно підключатися, а також додаткова службова інформація, така як логін доступу та пароль. Також на цьому екрані можемо завантажити перетворені дані у форматі CSV чи JSON (рис. 4.16):

Рисунок 4.16 – Екран експорту даних та доступу до інтерфейсів сховища даних

4.3 Висновки по розділу

Під час написання четвертого розділу ознайомилися із можливістю для розгортання програмного забезпечення, а також ознайомилися у деталях із можливостями взаємодії із системою за допомогою мобільного застосунка.

ВИСНОВКИ

У дипломному проекті було спроектоване розподілене сховище великих даних, яке може змінити свою внутрішню будову від запиту користувача, а також містить мобільний інтерфейс для управління сховищем. Програмне забезпечення можна використовувати для зберігання великих масивів інформації, а також зручно змінювати їх внутрішню структуру, коли необхідно виконувати інтелектуальну обробку та видавати через REST API вибіркові порції даних чи іншими методами взаємодіяти із стороннім програмним забезпеченням.

Для забезпечення роботоспроможності алгоритмів перетворення внутрішньої структури сховища, був використаний фреймворк для потокової роботи із великими даними на кластері Apache Spark. Загалом, у програмному забезпечені є можливість перетворити дані у формати Apache Hbase, Apache Phoenix, Parquet.

У першому розділі були розглянуті найбільш ефективні алгоритми і методи обробки великих обсягів даних для досягнення оптимального розподілу ресурсів у системі, а також певні види сховищ що краще виконують певні задачі, а також описано функціональні та нефункціональні вимоги до програмного продукту.

Під час написання другого розділу зроблений опис архітектури програмного забезпечення, за методологією IDEF0 були створені схеми для бізнес процесів, а також розібрані у деталях потрібні для правильної роботи програми функції та параметри.

Під час написання третього розділу проведено планування тестування програмного забезпечення сховища великих даних, що може змінювати свою внутрішню структуру у залежності від запиту користувача, що зможе виявити всі потенційні дефекти програмного забезпечення, які можуть призвести до небажаної поведінки системи в цілому та можливому погіршенню досвіду використання системи користувачами.

У четвертому розділі описано розгортання програмного забезпечення, а наведено можливості взаємодії із системою за допомогою мобільного додатку.

Результатом виконання дипломної роботи є функціонуючий мобільний додаток та серверна частина, які можна використовувати у будь якій сфері, де необхідна працювати із великими даними. У подальшому можливе розширення функціональності системи завдяки додаванню нових алгоритмів перетворення та структурування даних.

ПЕРЕЛІК ПОСИЛАНЬ

1) Apache Spark Docs [Електронний ресурс]: (Стаття) / Apache Spark Community – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://spark.apache.org/docs/latest/index.html. – Назва з екрана.

2) Apache Hbase Reference Guide [Електронний ресурс]: (Стаття) / Apache Hbase Community – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://hbase.apache.org/book.html. – Назва з екрана.

3) Apache Phoenix Grammar [Електронний ресурс]: (Стаття) / Apache Phoenix Community – Електрон. дан. (1 файл). – 2019. – Режим доступу: http://phoenix.apache.org/language/index.html. – Назва з екрана.

4) Apache Hadoop Docs [Електронний ресурс]: (Стаття) / Apache Hadoop Community – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://hadoop.apache.org/docs/stable/. – Назва з екрана.

5) Scala docs [Електронний ресурс]: (Стаття) / Scala Community – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://docs.scala-lang.org/. – Назва з екрана.

6) Akka Http docs [Електронний ресурс]: (Стаття) / Lightbend Inc – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://doc.akka.io/docs/akka-http/current/index.html. – Назва з екрана.

7) Akka Streams docs [Електронний ресурс]: (Стаття) / Lightbend Inc – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://doc.akka.io/docs/akka/current/stream/index.html. – Назва з екрана.

8) Postgres docs [Електронний ресурс]: (Стаття) / The PostgreSQL Global Development Group – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://www.postgresql.org/docs/. – Назва з екрана.

9) Yarn docs [Електронний ресурс]: (Стаття) / Yarn Community – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://classic.yarnpkg.com/en/docs/. – Назва з екрана.

10) Postman docs [Електронний ресурс]: (Стаття) / Postman Inc – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://learning.postman.com/docs/postman/api-documentation/documenting-your-api/. – Назва з екрана.

11) Airflow [Електронний ресурс]: (Стаття) / Apache Software Foundation – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://airflow.apache.org/. – Назва з екрана.

12) Bigtable [Електронний ресурс]: (Стаття) / Google Inc – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://cloud.google.com/bigtable. – Назва з екрана.

13) Rethink DB [Електронний ресурс]: (Стаття) / Rethink DB Community – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://rethinkdb.com/. – Назва з екрана.

14) Redis [Електронний ресурс]: (Стаття) / Redislabs – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://redis.io/. – Назва з екрана.

15) Kotlin [Електронний ресурс]: (Стаття) / Jetbrains – Електрон. дан. (1 файл). – 2020. – Режим доступу: https://kotlinlang.org/. – Назва з екрана.

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2020 р.

Розподілене сховище даних на основі Apache Spark
Опис програми
КПІ.ІП-6318.045430.02.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Ю.О. Олійник

Нормоконтроль:
_____ К.І. Ліщук

Виконавець:
_____ М.Д. Мамута

Київ – 2020 року

Тексти програмного коду

Розподілене сховище даних на основі Apache Spark

(Найменування програми (документа))

DVD-R

(Вид носія даних)

23 арк, 258 Кб

(Обсяг програми (документа) , арк.,) Кб)

Київ - 2020

Файл DFSReadWrite.scala

```
object DFSReadWrite {

  private var localFilePath: File = new File(".")
  private var dfsDirPath: String = ""

  private val NPARAMS = 2

  private def readFile(filename: String): List[String] = {
    val lineIter: Iterator[String] = fromFile(filename).getLines()
    val lineList: List[String] = lineIter.toList
    lineList
  }

  private def printUsage(): Unit = {
    val usage = """DFS Read-Write Test
    |Usage: localFile dfsDir
    |localFile - (string) local file to use in test
    |dfsDir - (string) DFS directory for read/write tests""".stripMargin

    println(usage)
  }

  private def parseArgs(args: Array[String]): Unit = {
    if (args.length != NPARAMS) {
      printUsage()
      System.exit(1)
```

```
  }

  var i = 0

  localFilePath = new File(args(i))
  if (!localFilePath.exists) {
    System.err.println(s"Given path (${args(i)}) does not exist")
    printUsage()
    System.exit(1)
  }

  if (!localFilePath.isFile) {
    System.err.println(s"Given path (${args(i)}) is not a file")
    printUsage()
    System.exit(1)
  }

  i += 1
  dfsDirPath = args(i)
}

def runLocalWordCount(fileContents: List[String]): Int = {
  fileContents.flatMap(_.split(" "))
    .flatMap(_.split("\t"))
    .filter(_.nonEmpty)
    .groupBy(w => w)
    .mapValues(_.size)
```

```
      .values
      .sum
  }


def main(args: Array[String]): Unit = {
    parseArgs(args)

    println("Performing local word count")
    val fileContents = readFile(localFilePath.toString())
    val localWordCount = runLocalWordCount(fileContents)

    println("Creating SparkSession")
    val spark = SparkSession
      .builder
      .appName("DFS Read Write Test")
      .getOrCreate()

    println("Writing local file to DFS")
    val dfsFilename = s"$dfsDirPath/dfs_read_write_test"

    // delete file if exists
    val fs = FileSystem.get(spark.sessionState.newHadoopConf())
    if (fs.exists(new Path(dfsFilename))) {
        fs.delete(new Path(dfsFilename), true)
    }
```

```
    val fileRDD = spark.sparkContext.parallelize(fileContents)
    fileRDD.saveAsTextFile(dfsFilename)

    println("Reading file from DFS and running Word Count")
    val readFileRDD = spark.sparkContext.textFile(dfsFilename)

    val dfsWordCount = readFileRDD
      .flatMap(_.split(" "))
      .flatMap(_.split("\t"))
      .filter(_.nonEmpty)
      .map(w => (w, 1))
      .countByKey()
      .values
      .sum

    spark.stop()
    if (localWordCount == dfsWordCount) {
      println(s"Success! Local Word Count $localWordCount and " +
        s"DFS Word Count $dfsWordCount agree.")
    } else {
      println(s"Failure! Local Word Count $localWordCount " +
        s"and DFS Word Count $dfsWordCount disagree.")
    }
  }
}
```

Файл SimpleTypedAggregator.scala

```
object SimpleTypedAggregator {

  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder
      .master("local")
      .appName("common typed aggregator implementations")
      .getOrCreate()

    import spark.implicits._
    val ds = spark.range(20).select(('id % 3).as("key"), 'id).as[(Long, Long)]
    println("input data:")
    ds.show()

    println("running typed sum:")
    ds.groupByKey(_._1).agg(new TypedSum[(Long, Long)](_._2).toColumn).show()

    println("running typed count:")
    ds.groupByKey(_._1).agg(new TypedCount[(Long, Long)](_._2).toColumn).show()

    println("running typed average:")
    ds.groupByKey(_._1).agg(new TypedAverage[(Long, Long)](_._2.toDouble).toColumn).show()

    println("running typed minimum:")
    ds.groupByKey(_._1).agg(new TypedMin[(Long, Long)](_._2.toDouble).toColumn).show()
```

```
    println("running typed maximum:")
    ds.groupByKey(_._1).agg(new TypedMax[(Long, Long)](_._2).toColumn).show()

    spark.stop()
  }
}
// scalastyle:on println

class TypedSum[IN](val f: IN => Long) extends Aggregator[IN, Long, Long] {
  override def zero: Long = 0L
  override def reduce(b: Long, a: IN): Long = b + f(a)
  override def merge(b1: Long, b2: Long): Long = b1 + b2
  override def finish(reduction: Long): Long = reduction

  override def bufferEncoder: Encoder[Long] = Encoders.scalaLong
  override def outputEncoder: Encoder[Long] = Encoders.scalaLong
}

class TypedCount[IN](val f: IN => Any) extends Aggregator[IN, Long, Long] {
  override def zero: Long = 0
  override def reduce(b: Long, a: IN): Long = {
    if (f(a) == null) b else b + 1
  }
  override def merge(b1: Long, b2: Long): Long = b1 + b2
  override def finish(reduction: Long): Long = reduction
```

```
  override def bufferEncoder: Encoder[Long] = Encoders.scalaLong
  override def outputEncoder: Encoder[Long] = Encoders.scalaLong
}

class TypedAverage[IN](val f: IN => Double) extends Aggregator[IN, (Double, Long), Double] {
  override def zero: (Double, Long) = (0.0, 0L)
  override def reduce(b: (Double, Long), a: IN): (Double, Long) = (f(a) + b._1, 1 + b._2)
  override def finish(reduction: (Double, Long)): Double = reduction._1 / reduction._2
  override def merge(b1: (Double, Long), b2: (Double, Long)): (Double, Long) = {
    (b1._1 + b2._1, b1._2 + b2._2)
  }

  override def bufferEncoder: Encoder[(Double, Long)] = {
    Encoders.tuple(Encoders.scalaDouble, Encoders.scalaLong)
  }
  override def outputEncoder: Encoder[Double] = Encoders.scalaDouble
}

class TypedMin[IN](val f: IN => Double) extends Aggregator[IN, MutableDouble, Option[Double]] {
  override def zero: MutableDouble = null
  override def reduce(b: MutableDouble, a: IN): MutableDouble = {
    if (b == null) {
      new MutableDouble(f(a))
    } else {
```

```
      b.value = math.min(b.value, f(a))
      b
    }
  }
  override def merge(b1: MutableDouble, b2: MutableDouble): MutableDouble
= {
    if (b1 == null) {
      b2
    } else if (b2 == null) {
      b1
    } else {
      b1.value = math.min(b1.value, b2.value)
      b1
    }
  }
  override def finish(reduction: MutableDouble): Option[Double] = {
    if (reduction != null) {
      Some(reduction.value)
    } else {
      None
    }
  }

  override def bufferEncoder: Encoder[MutableDouble] =
Encoders.kryo[MutableDouble]
  override def outputEncoder: Encoder[Option[Double]] =
Encoders.product[Option[Double]]
}
```

```
class TypedMax[IN](val f: IN => Long) extends Aggregator[IN, MutableLong,
Option[Long]] {
  override def zero: MutableLong = null
  override def reduce(b: MutableLong, a: IN): MutableLong = {
    if (b == null) {
      new MutableLong(f(a))
    } else {
      b.value = math.max(b.value, f(a))
      b
    }
  }
  override def merge(b1: MutableLong, b2: MutableLong): MutableLong = {
    if (b1 == null) {
      b2
    } else if (b2 == null) {
      b1
    } else {
      b1.value = math.max(b1.value, b2.value)
      b1
    }
  }
  override def finish(reduction: MutableLong): Option[Long] = {
    if (reduction != null) {
      Some(reduction.value)
    } else {
      None
    }
```

```
  }

  override def bufferEncoder: Encoder[MutableLong] =
Encoders.kryo[MutableLong]
  override def outputEncoder: Encoder[Option[Long]] =
Encoders.product[Option[Long]]
}

class MutableLong(var value: Long) extends Serializable

class MutableDouble(var value: Double) extends Serializable
```

Файл HbaseMain.scala

```
object HbaseMain {

  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.ALL)
    Logger.getLogger("akka").setLevel(Level.ALL)

    val conf = HBaseConfiguration.create()
    conf.addResource(new File("/Users/maxim/development/hbase-
1.4.9/conf/hbase-site.xml").toURI.toURL)

    val connection = ConnectionFactory.createConnection(conf)

    val admin = connection.getAdmin
    println("connection created")
```

```
    val newTable: TableName = TableName.valueOf("hbase_demo")
    val tDescriptor: HTableDescriptor = new HTableDescriptor(newTable)
    tDescriptor.addFamily(new
HColumnDescriptor("CF1").setCompressionType(Algorithm.NONE))
    createTableOrOverwrite(admin, tDescriptor)

    println("Table created")

    val table: Table = connection.getTable(newTable)

    insertLine(table, "1", "CF1", "name", "cesar")
    insertLines(table, "2", "CF1", Map("name" -> "kek", "age" -> "25"))

    println("Rows inserted")

    println("Reading created table...")
    printRows(getRows(table, Array("1", "2")))

    println("Search by name: cesar")
    scan(table, "CF1", "name", "cesar")

    connection.close()
    println("connection closed")
  }

}
```

Файл HbaseWriter.scala

```
object HBaseWriter {


  def createTableOrOverwrite(admin: Admin, table: HTableDescriptor): Unit =
{
    if (admin.tableExists(table.getTableName)) {
      admin.disableTable(table.getTableName)
      admin.deleteTable(table.getTableName)
    }
    admin.createTable(table)
  }


  def getRow(table: Table, rk: String): Result = {
    val get: Get = new Get(Bytes.toBytes(rk))
    table.get(get)
  }


  def getRows(table: Table, rks: Array[String]): Array[Result] = {
    val gets: Seq[Get] = rks.map(key => new Get(Bytes.toBytes(key)))
    table.get(gets)
  }


  def printRow(row: Result) = {
    val content: util.NavigableMap[Array[Byte], util.NavigableMap[Array[Byte],
Array[Byte]]] = row.getNoVersionMap
    for (entry <- content.entrySet) {
      for (sub_entry <- entry.getValue.entrySet) {
```

```
println(Bytes.toString(row.getRow) + "\t" + Bytes.toString(entry.getKey) + ":" + Bytes.toString(sub_entry.getKey) + " => " + Bytes.toString(sub_entry.getValue))
    }
   }
  }

  def printRows(rows: Array[Result]) = {
   rows.foreach(printRow(_))
  }

  def insertLine(table: Table, rk: String, cf: String, q: String, value: String): Unit = {
   val put = new Put(Bytes.toBytes(rk))
   put.addColumn(Bytes.toBytes(cf), Bytes.toBytes(q), Bytes.toBytes(value))
   table.put(put)
  }

  def insertLines(table: Table, rk: String, cf: String, content: Map[String, String]): Unit = {
   val put = new Put(Bytes.toBytes(rk))
   content.foreach(x => put.addColumn(Bytes.toBytes(cf), Bytes.toBytes(x._1), Bytes.toBytes(x._2)))
   table.put(put)
  }

  def scan(table: Table, cf: String, cn: String, value: String): Unit = {
   val scan = new Scan()
```

```
    val filter = new SingleColumnValueFilter(Bytes.toBytes(cf), Bytes.toBytes(cn), CompareOp.EQUAL, Bytes.toBytes(value))
    scan.setFilter(filter)

    val scanner = table.getScanner(scan)
    val result = scanner.iterator()
    while(result.hasNext)
    {
      val data = result.next()
      val name = Bytes.toString(data.getValue(Bytes.toBytes(cf),Bytes.toBytes(cn)))

      println()
      println("name::"+name)
    }
  }
}
```

Файл PhoenixMain.scala

```
object PhoenixMain {

  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.ALL)
    Logger.getLogger("akka").setLevel(Level.ALL)

    val conf = HBaseConfiguration.create()
    conf.addResource(new File("/Users/maxim/development/hbase-1.4.9/conf/hbase-site.xml").toURI.toURL)
```

```
    val connection = ConnectionFactory.createConnection(conf)

    val admin = connection.getAdmin
    println("connection created")

    val newTable: TableName = TableName.valueOf("hbase_demo")
    val tDescriptor: HTableDescriptor = new HTableDescriptor(newTable)
    tDescriptor.addFamily(new
HColumnDescriptor("CF1").setCompressionType(Algorithm.NONE))
    createTableOrOverwrite(admin, tDescriptor)

    println("Table created")

    val table: Table = connection.getTable(newTable)

    insertLine(table, "1", "CF1", "name", "cesar")
    insertLines(table, "2", "CF1", Map("name" -> "kek", "age" -> "25"))

    println("Rows inserted")

    println("Reading created table...")
    printRows(getRows(table, Array("1", "2")))

    println("Search by name: cesar")
    scan(table, "CF1", "name", "cesar")

    connection.close()
```

```
    println("connection closed")
  }

}
case class Record(key: Int, value: String)

object RDDRelation {
  def main(args: Array[String]): Unit = {
    // $example on:init_session$
    val spark = SparkSession
      .builder
      .appName("Spark Examples")
      .config("spark.some.config.option", "some-value")
      .getOrCreate()

    // Importing the SparkSession gives access to all the SQL functions and implicit conversions.
    import spark.implicits._
    // $example off:init_session$

    val df = spark.createDataFrame((1 to 100).map(i => Record(i, s"val_$i")))
    // Any RDD containing case classes can be used to create a temporary view. The schema of the
    // view is automatically inferred using scala reflection.
    df.createOrReplaceTempView("records")

    // Once tables have been registered, you can run SQL queries over them.
    println("Result of SELECT *:")
```

```
spark.sql("SELECT * FROM records").collect().foreach(println)

// Aggregation queries are also supported.
val count = spark.sql("SELECT COUNT(*) FROM records").collect().head.getLong(0)
println(s"COUNT(*): $count")

// The results of SQL queries are themselves RDDs and support all normal RDD functions. The
// items in the RDD are of type Row, which allows you to access each column by ordinal.
val rddFromSql = spark.sql("SELECT key, value FROM records WHERE key < 10")

println("Result of RDD.map:")
rddFromSql.rdd.map(row => s"Key: ${row(0)}, Value: ${row(1)}").collect().foreach(println)

// Queries can also be written using a LINQ-like Scala DSL.
df.where($"key" === 1).orderBy($"value".asc).select($"key").collect().foreach(println)

// Write out an RDD as a parquet file with overwrite mode.
df.write.mode(SaveMode.Overwrite).parquet("pair.parquet")

// Read in parquet file.  Parquet files are self-describing so the schema is preserved.
val parquetFile = spark.read.parquet("pair.parquet")
```

```
    // Queries can be run using the DSL on parquet files just like the original RDD.
    parquetFile.where($"key" === 1).select($"value".as("a")).collect().foreach(println)

    // These files can also be used to create a temporary view.
    parquetFile.createOrReplaceTempView("parquetFile")
    spark.sql("SELECT * FROM parquetFile").collect().foreach(println)

    spark.stop()
  }
}
```

Файл StartLocal.scala

```
object Main {

  //spark-sql read from csv
  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.ERROR)
    Logger.getLogger("akka").setLevel(Level.ERROR)

    val spark = SparkSession.builder
      .master("local")
      .appName("Spark CSV Reader")
      .getOrCreate

    readLocalCSVFileAndWriteLocal(spark)
    readHDFSCSVFile(spark)
```

```
    readLocalParquet(spark)
    readHDFSParquet(spark)

    spark.stop()
  }

  private def readLocalCSVFileAndWriteLocal(spark: SparkSession): Unit = {
    //read csv file locally
    val df = spark.read.format("csv").option("header", "true").load("res/test.csv")

    println()
    println("Local CSV file: ")
    println()

    df.show()
    // Print the schema in a tree format
    df.printSchema()

    //write to
    df.write.mode(SaveMode.Overwrite).format("parquet").parquet("test")
  }

  private def readHDFSCSVFile(spark: SparkSession): Unit = {
    //read csv file remotely
    val df = spark.read.format("csv").option("header",
"true").load("hdfs://localhost:9000/res/test.csv")

    println()
```

```
    println("HDFS CSV file: ")
    println()

    df.show()
    // Print the schema in a tree format
    df.printSchema()

    //write to

df.write.mode(SaveMode.Overwrite).format("parquet").parquet("hdfs://localhos
t:9000/parquet/test")
  }

  private def readLocalParquet(spark: SparkSession): Unit = {
    //read local parquet file
    val df = spark.read.format("parquet").load("test")

    println()
    println("Local parquet file: ")
    println()

    df.show()
    // Print the schema in a tree format
    df.printSchema()
  }

  private def readHDFSParquet(spark: SparkSession): Unit = {
    //read hdfs parquet file
```

```
    val df = spark.read.format("parquet").load("hdfs://localhost:9000/parquet/test")

    println()
    println("HDFS parquet file: ")
    println()

    df.show()
    // Print the schema in a tree format
    df.printSchema()
  }

}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Розподілене сховище даних на основі Apache Spark

Технічне завдання

КПІ.ІП-6318.045430.03.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____М.Д. Мамута

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки:

Галузь застосування:

Наведене технічне завдання поширюється на розробку серверного програмного забезпечення та мобільного додатку для платформи Android “Розподілене сховище даних на основі Apache Spark”, котра використовується для перетворення внутрішньої структури сховища даних та призначена для оптимального зберігання великих даних.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки сховища великих даних є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для перетворення внутрішньої структури сховища великих даних.

Метою розробки збільшення можливості сховищ даних за рахунок організації внутрішньої структури у залежності від запитів користувача .

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Вимоги до функціональних характеристик

4.1.1. Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1. Для користувача:

- при вході у мобільний додаток відображати екран із списком методів введення великих даних;
- наявність екранів із поясненнями як ввести дані у систему сторонніми методами;
- екран для завантаження даних у форматі csv;
- головний екран із вибором методів внутрішнього перетворення сховища даних;
- відображення процесу перетворення сховища і відображення результату та можливих помилок при перетворенні;
- відображення екрану із доступними інтерфейсами доступу до сховища даних сторонніми засобами;
- можливість завантажити перетворені файли у форматі json та csv .

4.1.2. Розробку виконати на платформі macOS Catalina.

4.2. Вимоги до надійності

4.2.1. Передбачити контроль введення великих даних за допомогою csv, json файлів та терміналу, виконання методів перетворення завдяки кнопкам користувацького інтерфейсу.

4.2.2. Передбачити захист від некоректних дій користувача шляхом блокування кнопок після початку процесу перетворення сховища до моменту отримання результату.

4.2.3. Забезпечити цілісність допоміжних даних системи, а також безпосередньо процесу внутрішнього перетворення сховища великих даних.

4.3. Умови експлуатації

4.3.1. Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.2. Вимоги до обслуговування не виставляються

4.3.3. Вимоги до обслуговуючого персоналу не виставляються

4.4. Вимоги до складу і параметрів технічних засобів

4.4.1. Програмне забезпечення повинно функціонувати на мобільних телефонах з системою Android, серверне програмне забезпечення має функціонувати на комп’ютерах із операційною системою Linux

4.4.2. Мінімальна конфігурація технічних засобів:

4.4.2.1. Тип процессору серверу ..... Intel Core i3.

4.4.2.2. Об‘єм ОЗП серверу ..... 2048 Мб.

4.4.2.3 Об‘єм ОЗП телефону ……. 512 Мб.

4.5. Вимоги до інформаційної та програмної сумісності

4.5.1. Програмне забезпечення серверу повинно працювати під управлінням операційних систем сімейства Linux (Ubuntu, Debian і т.д.) або Unix, мобільний додаток має працювати під управлінням системи Android.

4.5.2. Вхідні дані повинні бути представлені в наступному форматі: csv файл, JSON файл, POST HTTP запит із тілом типу json

4.5.3. Результати повинні бути представлені в наступному форматі: JSON файл, csv файл, інтерфейси доступу до баз даних обраного типу сховища.

4.5.4. Програмне забезпечення серверу повинно бути створеним із використанням фреймворку Apache Spark і написано на мові програмування Scala, мобільний додаток має бути написаний із використанням Android SDK та мови програмування Kotlin.

4.6. Вимоги до маркування та пакування

Вимоги до маркування та пакування не пред‘являються.

4.7. Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не пред'являються.

4.8. Спеціальні вимоги

Згенерувати установчу версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1. Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

5.2. Програмне забезпечення повинно мати довідникову систему

5.3. У склад супроводжувальної документації повинні входити наступні документи:

5.3.1. Пояснювальна записка не менше ніж на 100 аркушах формату А4 (без додатків 5.3.2 - 5.3.6).

5.3.2. Технічне завдання.

5.3.3. Керівництво користувача.

5.3.4. Керівництво системного програміста

5.3.5. Керівництво адміністратора

5.3.6. Програма та методика тестування

5.4. Графічна частина повинна бути виконана на аркушах формату А3, котрі включаються у якості додатків до пояснювальної записки:

5.4.1. Схема структурна варіантів використань

5.4.2. Схема структурна бізнес процесів

5.4.3. Креслення вигляду екранних форм

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк,	Звітність
1.	Вивчення літератури за тематикою проєкту	15.04.2020	
2.	Розробка технічного завдання	21.04.2020	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	18.04.2020	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	24.04.2020	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму …)
5.	Програмна реалізація програмного забезпечення	24.04.2020	Тексти програмного забезпечення
6.	Тестування програмного	24.04.2020	Тести, результати

	забезпечення		тестування
7.	Розробка матеріалів текстової частини проєкту	01.05.2020	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проєкту	10.05.2020	Графічний матеріал проєкту
9.	Оформлення технічної документації проєкту	15.05.2020	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1. Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Розподілене сховище даних на основі Apache Spark

Програма та методика тестування

КПІ.ІП-6318. 045430.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ М.Д. Мамута

Київ – 2020 року

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Розподілене сховище даних на основі Apache Spark представляє собою сервіс, створений на основі Apache Spark фреймворку та Hadoop HDFS із використанням мови програмування Scala та технології Akka Http, а також клієнтський мобільний додаток, створений для платформи Android із використанням мови програмування Kotlin.

2 МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

- функціональна працездатність елементів екранів мобільного застосунку;
- забезпечення доступу до введення великих даних до сховища;
- забезпечення належного рівня безпеки даних;
- забезпечення перетворення внутрішньої будови сховища даних;
- наявність доступу до бази даних розподіленого сховища;
- відповідність дизайну вимогам Технічного завдання.

3 МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

- функціональне тестування, зокрема на рівні Critical path test (ба-зове тестування);
- тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
- тестування інтерфейсу.

Маємо протестувати кожний можливий варіант (test case), як ще кажуть, прецедент. Результат кожного тесту та інформація про його перебіг буде збережена до сервісу. Людина, що тестує, проводить кожний із тестів і позначає результат його виконання за кожною із умов.

Після того, як тестування було завершено, буде проведено перегляд звіту із тестування. У разі знаходження будь-якої із помилок - їх перелік передається розробнику сервісу для їх подальшого виправлення.

Основна функціональність системи та всі її складові мають функціонувати як очікувалося та бути окресленими в окремих випадках тестування. Не повинно бути виявлено критичних дефектів, тобто дефектів, при яких основна функціональність системи не виконує, або тільки частково виконує свою задачу.

Кінцевий користувач має мати змогу успішно вибрати метод введення великих даних, ввести їх, вибрати алгоритм внутрішнього перетворення сховища, виконати алгоритм внутрішнього перетворення сховища, побачити результат виконання чи помилку, побачити список доступних інтерфейсів перетвореного сховища даних, завантажити перетворені дані одним методів.

Будуть використані наступні методи тестування: функціональне, тестування системної продуктивності, тестування користувацького інтерфейсу та системи в цілому. Тестування буде виконуватися завдяки інструментам Unit test та Scala test.

Не менше ніж 90% від усіх тестових випадків мають бути успішно пройдені. Жоден із невдало пройдених випадків тестування не повинен мати вирішального значення для можливостей кінцевого користувача використовувати систему.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію SpecFlow.

Працездатність web-ресурсу перевіряється шляхом:

- динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
- динамічного ручного тестування на відповідність функціональним вимогам;
- статичного тестування коду;
- тестування мобільного додатку на різних Android пристроях;
- тестування при максимальному навантаженні;
- тестування стабільності роботи при різних умовах;
- тестування зручності використання;
- тестування інтерфейсу.

Під час тестування має бути перевірена уся функціональна складова програмного забезпечення. Функції програмного забезпечення, що підлягають тестуванню:

- вибір методу введення великих даних;
- додавання CSV файлу із локальної файлової системи до віддаленої;
- вибір алгоритму перетворення внутрішньої будови сховища;
- демонстрація результатів перетворення чи помилки;
- демонстрація інтерфейсів доступу до перетвореного сховища;
- завантаження CSV файлу із перетвореними даними;
- завантаження JSON файлу із перетвореними даними.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Розподілене сховище даних на основі Apache Spark

Керівництво користувача

КПІ.ІП-6318. 045430.05.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ М.Д. Мамута

Київ – 2020 року

ЗМІСТ

1 ВВЕДЕННЯ ДАНИХ

Для того, щоб почати роботу із нашим сервісом, необхідно запустити додаток на будь-якому Android телефоні із версією ОС 5+. Після встановлення apk додатку на телефон, запускаємо його і потрапляємо на головну сторінку додатку (Рисунок 4.1):

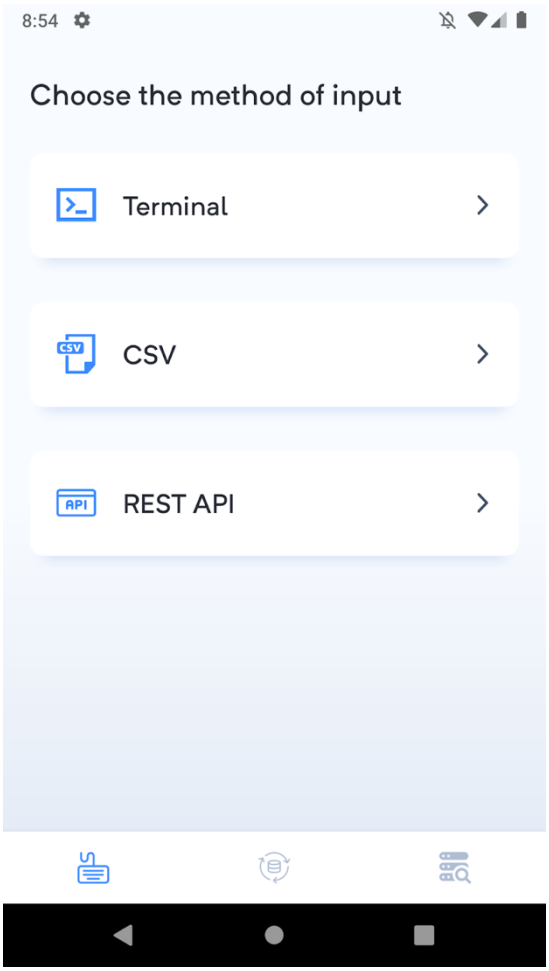


Рисунок 1 – Головний екран додатку

На цьому екрані можемо обрати один із доступних варіантів завантаження великих даних у систему: через термінал, CSV файл або за допомогою REST API. Якщо виберемо варіант термінального введення, то перейдемо на сторінку із детальним описом, як саме можемо ввести дані. Також якщо перейдемо на екран із методів введення через REST API, то також отримаємо детальну інструкцію, як можемо це зробити, ці два методи зображені на (Рисунок 2):

Рисунок 2 – Екрани допомоги із термінальним та REST API методами введення даних

Якщо виберемо варіант із методів введення це CSV файл, то отримаємо можливість прикріпити CSV файл із внутрішньої файлової системи мобільного телефону. Для того, щоб це виконати, необхідно натиснути на картинку “прикріпити”, обрати необхідний CSV файл у системі та очікувати його завантаження. Екран, де можемо прикріпити CSV файл і завантажити його до системи зображений на (Рисунок 3):

Рисунок 3 – Екран прикріплення CSV файлу

2 ВИБІР АЛГОРИТМУ

Далі можемо перейти до методів внутрішнього перетворення структури сховища (центральна вкладка у навігаційному меню). На цьому екрані можемо обрати на вибір один із методів внутрішнього перетворення сховища, такі як: Hbase, що є дуже ефективним для вибіркового пошуку даних у сховищі, Phoenix, який можна використовувати із SQL операціями та Parquet, який є оптимальним варіантом зберігання даних для системи, яка працює із Apache Spark. Далі можемо обрати один із можливих джерел даних у нашому сховищі, такі як: CSV, files та JSON.

3 ВИКОНАННЯ АЛГОРИТМУ

Далі можемо натиснути на кнопку перетворення та процес розпочнеться. Цей екран зображено на (Рисунок 4):

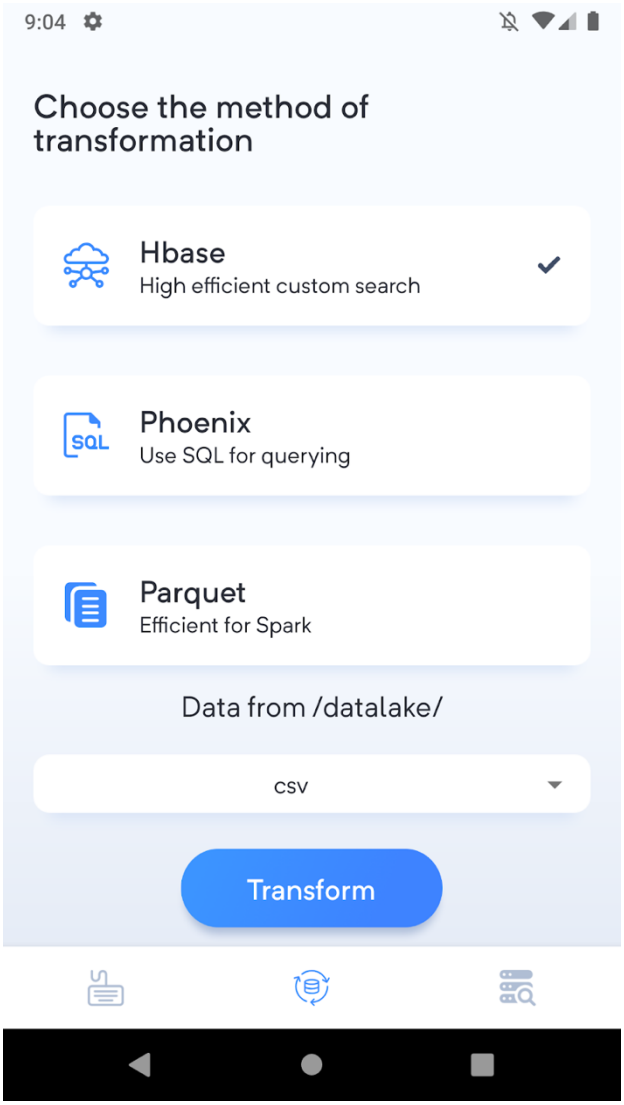


Рисунок 4 – Екран вибору методу перетворення

4 ПЕРЕВІРКА ВИКОНАННЯ ТА ДОСТУПУ ДО СХОВИЩА

Далі перейдемо на екран, де зображені методи доступу до перетворених даних сховища, а саме інтерфейси доступу до Hbase, Phoenix чи Parquet файлів. У цих інтерфейсах зображені порт, до якого необхідно підключатися, а також додаткова службова інформація, така як логін доступу та пароль. Також на цьому екрані можемо завантажити перетворені дані у форматі CSV чи JSON (Рисунок 5):

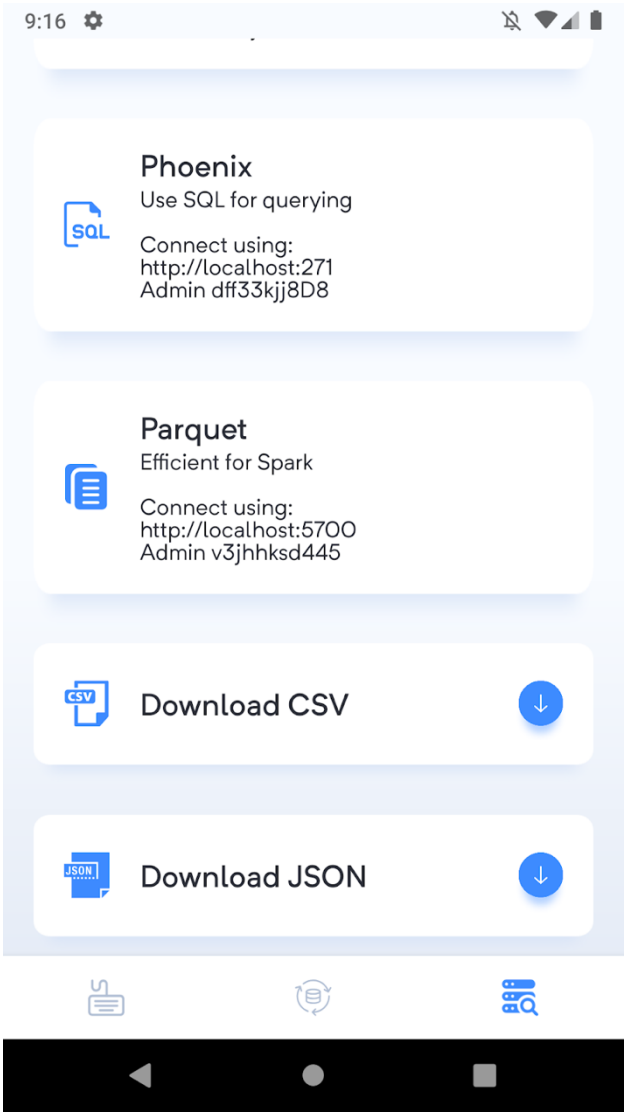


Рисунок 5 – Екран експорту даних та доступу до інтерфейсів сховища даних

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

Розподілене сховище даних на основі Apache Spark

Графічні матеріали

КПІ.ІП-6318. 045430.06.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Ю.О. Олійник

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ М.Д. Мамута

Київ – 2020 року

КПІ.ІП-6318.045430.06.99.СС

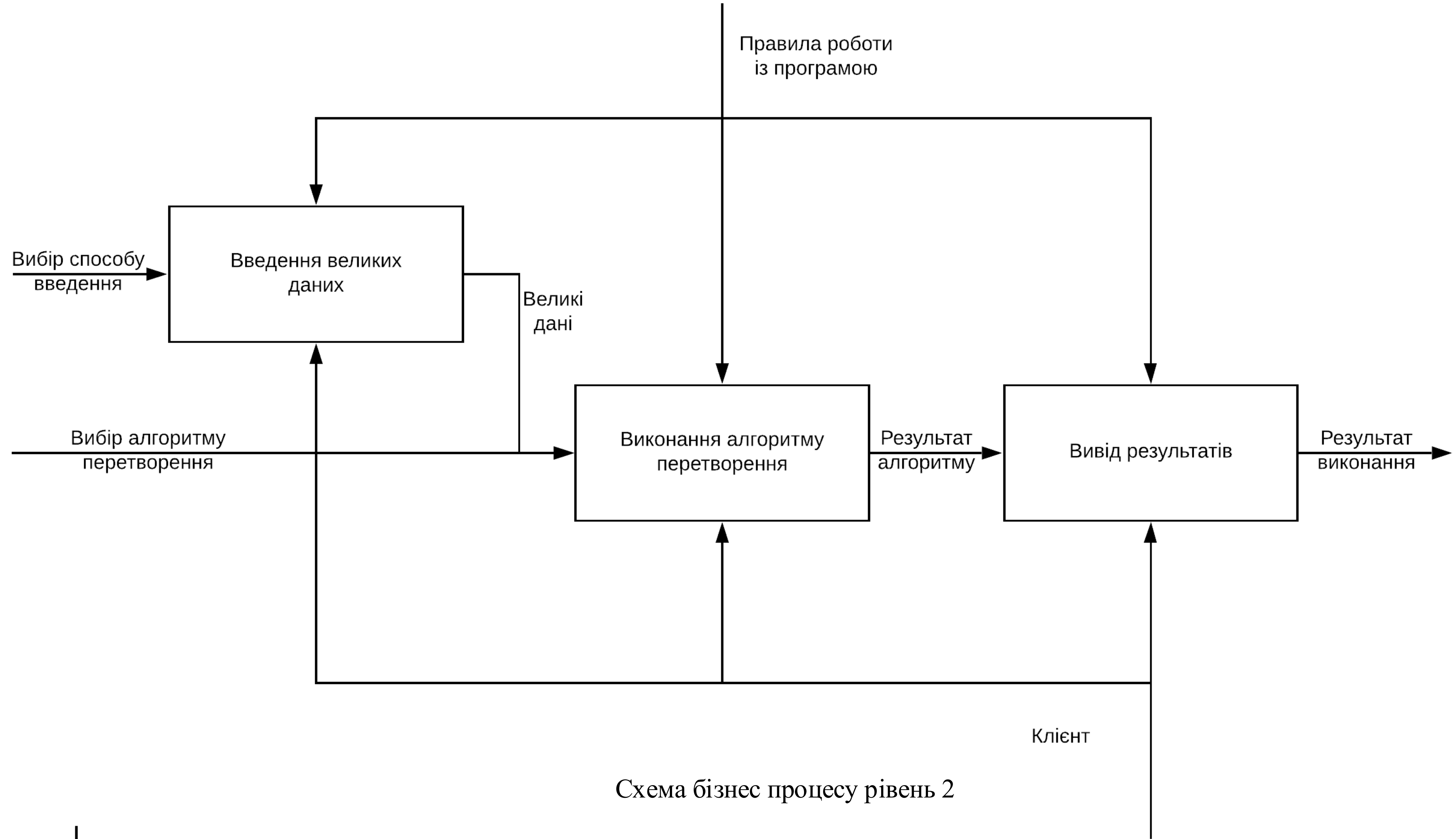


Схема бізнес процесу рівень 2

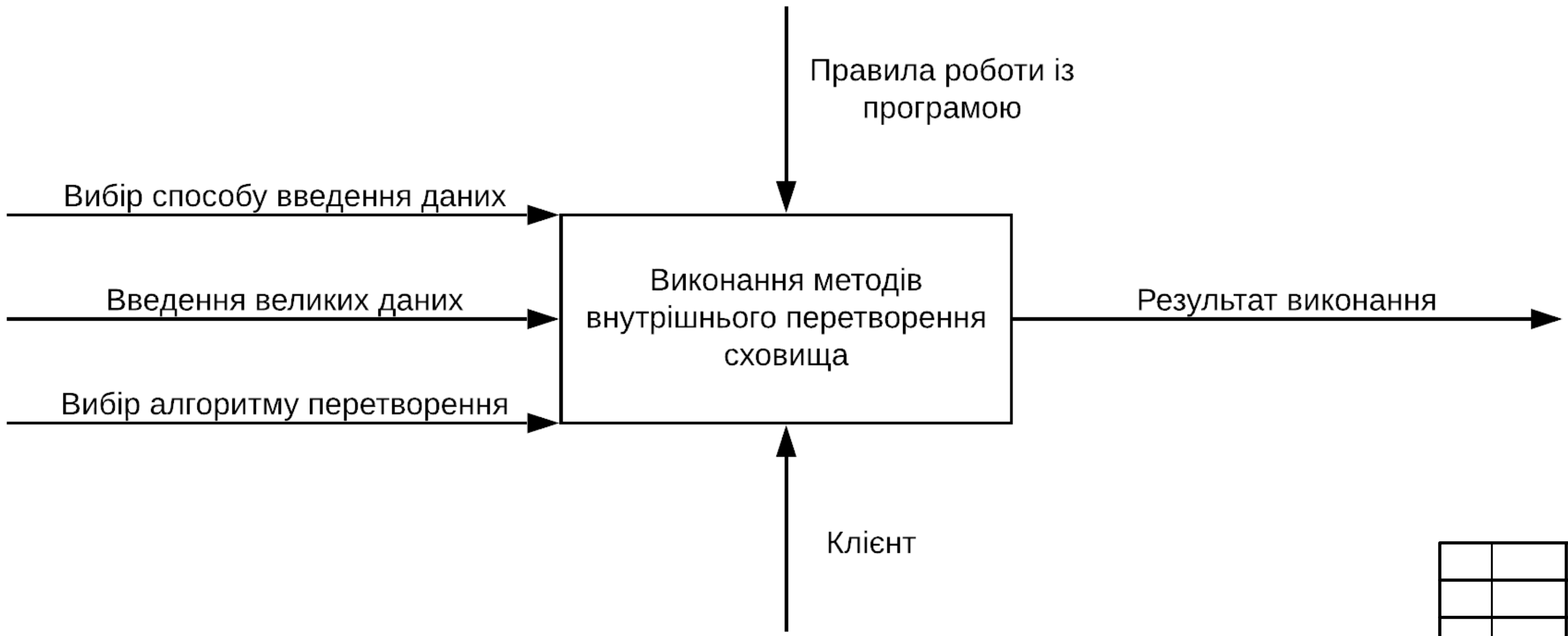


Схема бізнес процесу рівень 1

					КПІ.ІП-6318.045430.06.99.СС			
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Мамута М.Д.			Схема структурна бізнес процесів	Літера	Маса	Масштаб
Перевірив		Олійник Ю.О.						
Т. кон.						Аркуш 1	Аркушів 1	
					Розподілене сховище даних на основі Apache Spark	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Н. кон.		Ліщук К.І.						
Затвердив		Олійник Ю.О.						

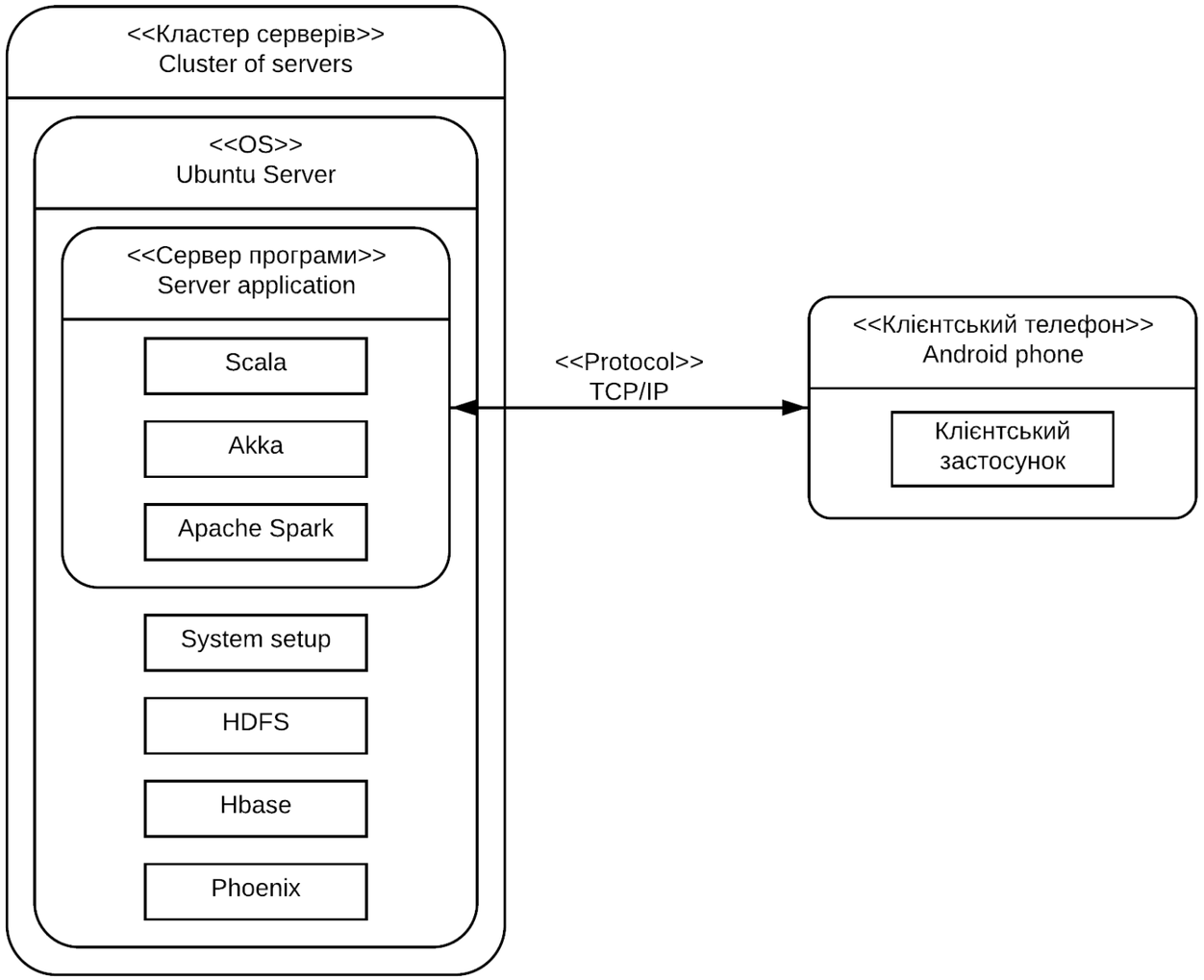
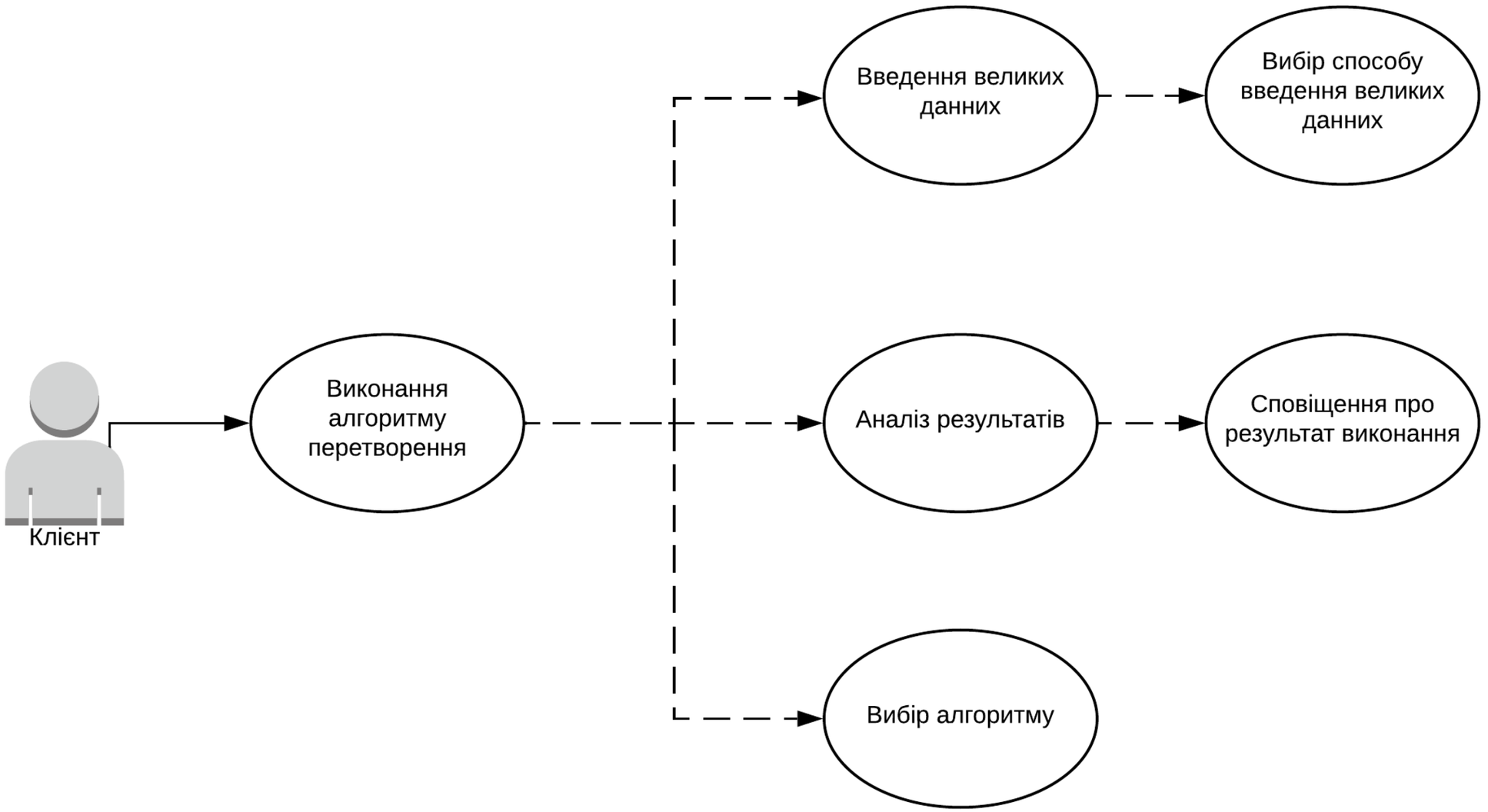


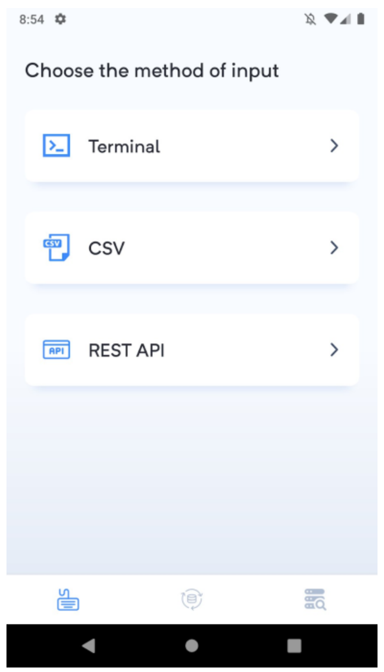
Схема бізнес процесу

					КПІ.ІП-6318.045430.06.99.СС			
						Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата	Схема структурна розгортання			
Розробив		Мамута М.Д.				Аркуш 1	Аркушів 1	
Перевірив		Олійник Ю.О.						
Т. кон.								
					Розподілене сховище даних на основі Apache Spark	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Н. кон.		Ліщук К.І.						
Затвердив		Олійник Ю.О.						

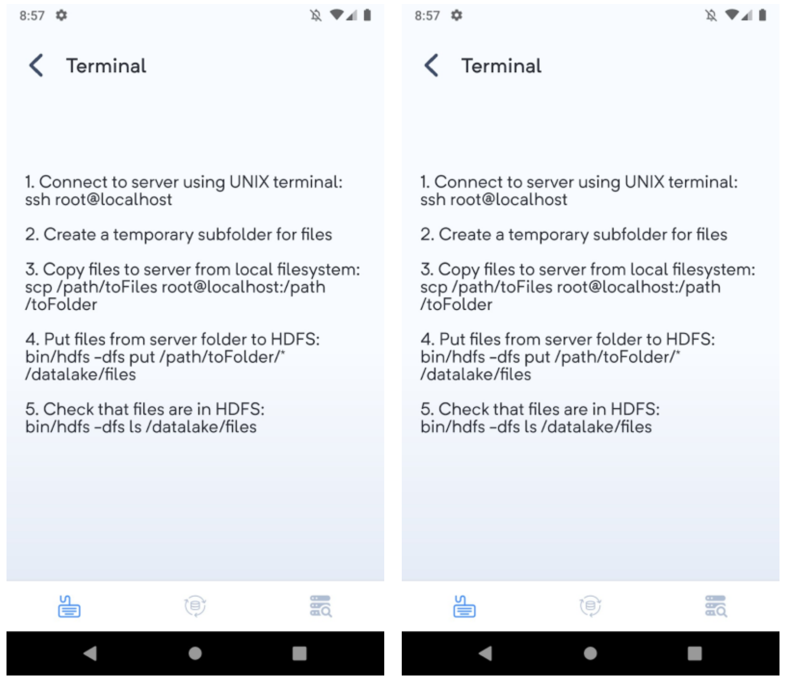


					КПІ.ІП-6318.045430.06.99.СС			
					Схема структурна варіантів використання	Літера	Маса	Масштаб
Зм.	Арк.	№ документа	Підпис	Дата				
Розробив		Мамута М.Д.				Аркуш 1	Аркушів 1	
Перевірив		Олійник Ю.О.						
Т. кон.								
					Розподілене сховище даних на основі Apache Spark	КПІ ім.Ігоря Сікорського Кафедра АСОІУ гр. ІП-63		
Н. кон.		Ліщук К.І.						
Затвердив		Олійник Ю.О.						

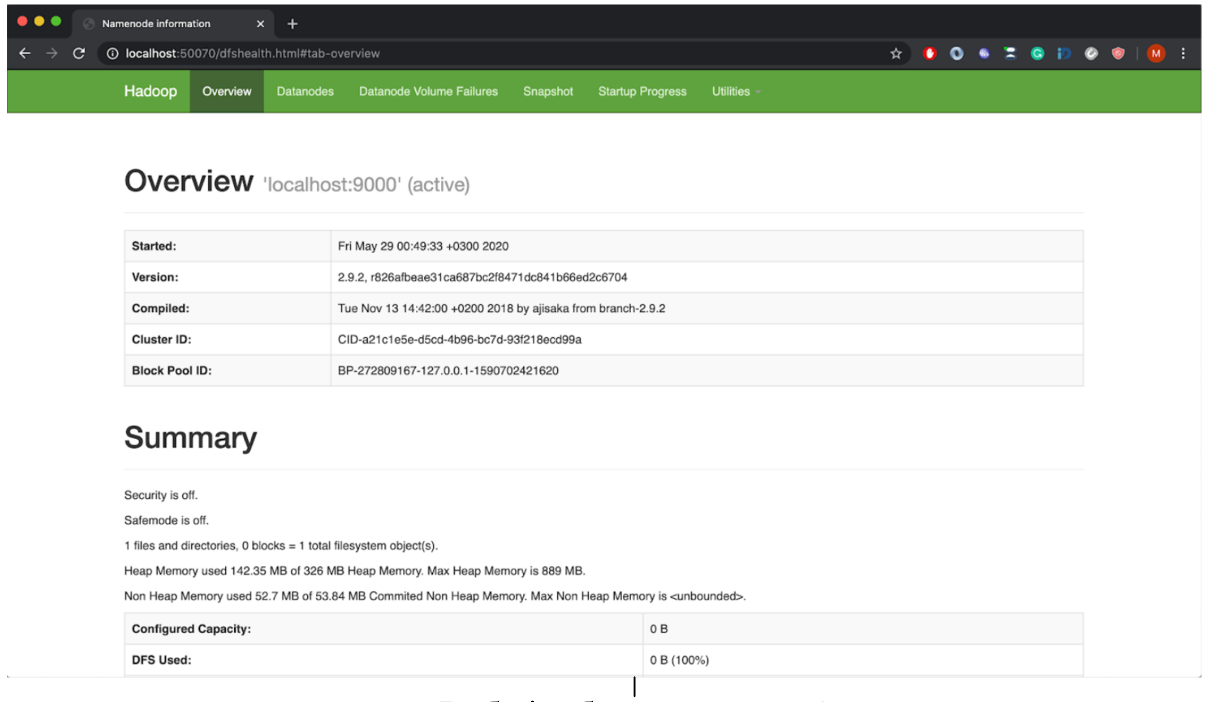
КПІ.ІП-6318.045430.06.99.КЕ



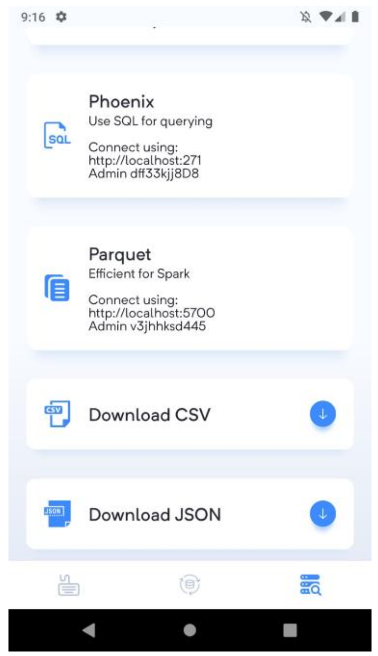
Головний екран застосунку



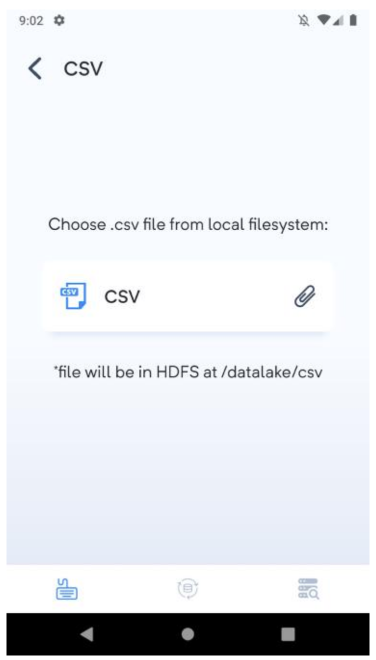
Екрани допомоги із термінальним та REST API методами введення даних



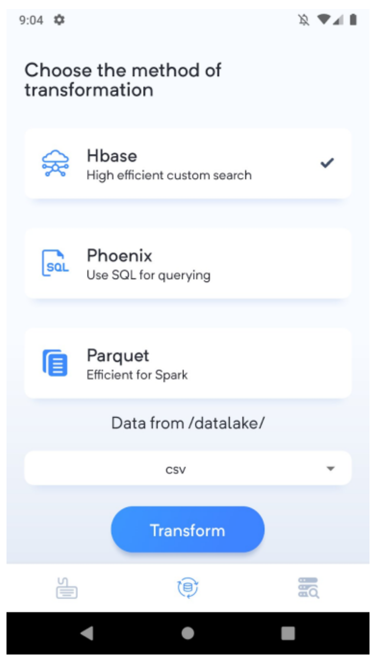
Веб відображення HDFS



Екран експорту даних та доступу до інтерфейсів сховища даних



Екран прикріплення CSV файлу



Екран вибору методу перетворення

					КПІ.ІП-6318.045430.06.99.КЕ
Зм.	Арк.	№ документа	Підпис	Дата	
Розробив		Мамута М.Д.			Креслення вигляду екранних форм
Перевірив		Олійник Ю.О.			
Т. кон.					
					Розподілене сховище даних на основі Apache Spark
Н. кон.		Ліщук К.І.			
Затвердив		Олійник Ю.О.			

Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

КПІ ім.Ігоря Сікорського
Кафедра АСОІУ
гр. ІП-63